

mBot and Me

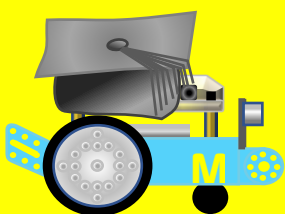
(or to be more accurate, mBot and Us)

by Lindsay Rooms MBE



An informative guide for both the young and the not so young !

An explanation of everything mBot; including how to use the mBlock 5 programming interface, full details of useful add-on components and the best ways to modify your new robot.



Contains a comprehensive and in-depth guide to designing and scripting programmes using mBlock 5.

● Windows 10 / mBlock v. 5.1.0 - 2020 Edition

mBot and Me

(or to be more accurate, mBot and us)

by Lindsay Rooms MBE

**An in-depth guide
for both the young
and the not so young !**

*A comprehensive and sequential explanation of everything mBot;
including how to use the mBlock 5 programming interface, full
details of useful add-on components and the best ways to modify
and programme your new robot.*

Windows 10 / mBlock 5

2020 Edition

For **Emma**

What will she become...

... a Scientist ?

... a Gymnast ?

... an Engineer ?

... a Dancer ?

... an Astronaut ?

Que Sera, Sera - *'Whatever Will Be, Will Be'*.

Whatever she does, I will be a proud Grandad (and it's been a privilege).

"Don't let your past dictate who you are; but let it be part of who you will become".

also dedicated to

John Coll

Pioneer P.C. Specialist

‡ I.T. Philanthropist

'A valued colleague from the 1970's who encouraged me to programme computers'

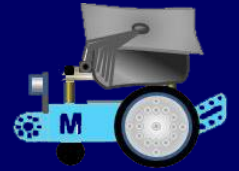
John Coll was asked by the BBC to help draw up the functional description for a computer which could be used as part of a television series to teach computer literacy. - Quote: "It was John's drive, determination and sheer brilliance that really pulled the whole thing off".

Contents

Chapter 1	An Introduction (... or how it all began ...)	1
Chapter 2	About Makeblock, Scratch and Arduino	4
Chapter 3	About mBot the Robotics Device	8
Chapter 4	About mBlock 5	11
Chapter 5	About the mBlock 5 User Interface	15
Chapter 6	Setting up and Connecting mBot	22
Chapter 7	About mBot's Remote Control Modes	28
Chapter 8	All About Batteries	33
Chapter 9	mBlock 5 / Scratch - the Basics	35
Chapter 10	About mBlock 5 in more Detail	44
Chapter 11	Programming with mBlock 5	52
Chapter 12	mBlock 5 - Graphics Programming	70
Chapter 13	Discovering Broadcast Messages	76
Chapter 14	Creating Graphics Libraries for mBlock 5	80
Chapter 15	Building a Control Interface for mBot	98
Chapter 16	Creating ' Smart Systems ' in mBlock 5	137
Chapter 17	About Good Habits and Best Practice	151
Chapter 18	Quo Vadis?	157
Chapter 19	An mBot ' Radar ' Simulation Project	160
Chapter 20	An mBot Drawing Machine Project	171

Appendices

Appendix 1	mBot ‘add-on’ Component - LED Panel	184
Appendix 2	mBot ‘add-on’ Component - Servo Pack	192
Appendix 3	mBot Servo Project - Robot - ‘ <i>Dancing Cat</i> ’	199
Appendix 4	mBot Servo Project - Robot - ‘ <i>Head-Shaking Cat</i> ’	205
Appendix 5	mBot Servo Project - Robot - ‘ <i>Light-Emitting Cat</i> ’	215
Appendix 6	mBot ‘add-on’ Component - Six Legged Robot Pack	219
Appendix 7	mBot ‘Walker’ Project - Robot ‘ <i>Beetle</i> ’	221
Appendix 8	mBot ‘Walker’ Project - Robot ‘ <i>Mantis</i> ’	226
Appendix 9	mBot ‘Walker’ Project - Robot ‘ <i>Crazy Frog</i> ’	229
Appendix 10	mBot ‘add-on’ Component - Light & Sound Pack	231
Appendix 11	mBot & Mblock 5 - All you need to know about LEDs	233
Appendix 12	mBot Light & Sound Project - Robot ‘ <i>Light-Chasing</i> ’	239
Appendix 13	mBot Light & Sound Project - ‘ <i>Intelligent Desk Light</i> ’	243
Appendix 14	mBot Light & Sound Project - Robot ‘ <i>Scorpion</i> ’	253
Appendix 15	mBot components - the latest bits	256



Chapter 1 - An Introduction (... or how it all began...)

Well it all began on Christmas morning, December 2017...

... After breakfast was over, my granddaughter Emma (aged 7 1/4) began to demolish the *huge* pile of presents under Grandma & Grandad's Christmas Tree - eventually we could see the floor and one neat & medium-sized (almost 200mm square & 100mm deep) box-like present remained. It was for me! It felt fairly solid and fairly heavy and when I unwrapped it, the box said '*Makeblock - construct your dreams*' and '*mBot*'. It also said that was an '*Educational*' robot construction kit.



It also said on the front of the little booklet inside the top of the box that it was '*mBot - One Robot per Child*' for beginners to learn **STEM** (Science, Technology, Engineering & Mathematics) - so was this really a present for me? Emma's mum (my daughter) explained quickly that this was for me to show Emma how to build & modify the robot and how to programme it" ... it works on Scratch you know" and "... its Arduino you know!". "... Arduino - is that a fundamental particle of nuclear physics?" - I was now out of my depth, I hadn't a clue.

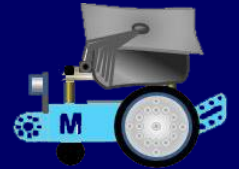
Had she thought to herself - Emma's mum, that is - "That'll keep the old fossil's (or something ruder perhaps?) brain working..."?"

Keeping the old-boy's brain overloaded more like! Well, it is my seventieth birthday this year.

I had heard of Scratch and knew that it was a graphical programming interface for kids using coloured blocks that fit together like a sequential jigsaw, but I had never used it and had considered it to be a bit too simple. I wondered how much mBot had cost them? Ball-park figure in the UK about £85 I guess.

So, with the festivities in full swing & lunch imminent, I had to put the box to one side. Eventually, in a lull in late afternoon, Emma and I 'unboxed' it (that's a fashionable consumer word isn't it). The processor board and the chassis were in full view at the top of the box in a very neat cardboard tray. We lifted it out and examined the contents - it was all beautifully made. In the bottom of the box was another cardboard divided tray with all the other bits. Little brown plastic bags that contained mysterious circuit boards which we left untouched, bags of screws & nuts, cables & the wheels. Emma found the Screwdriver (she's so practical) and had whipped-off the wrapping before I could blink and was exclaiming about it being double ended "... it's like an Allan key shape" she said meaning the hexagonal cross-section. She studied all the bits that were unpacked and tea-time approaching, I promised her that we (she, I mean) would build it soon - but not today.

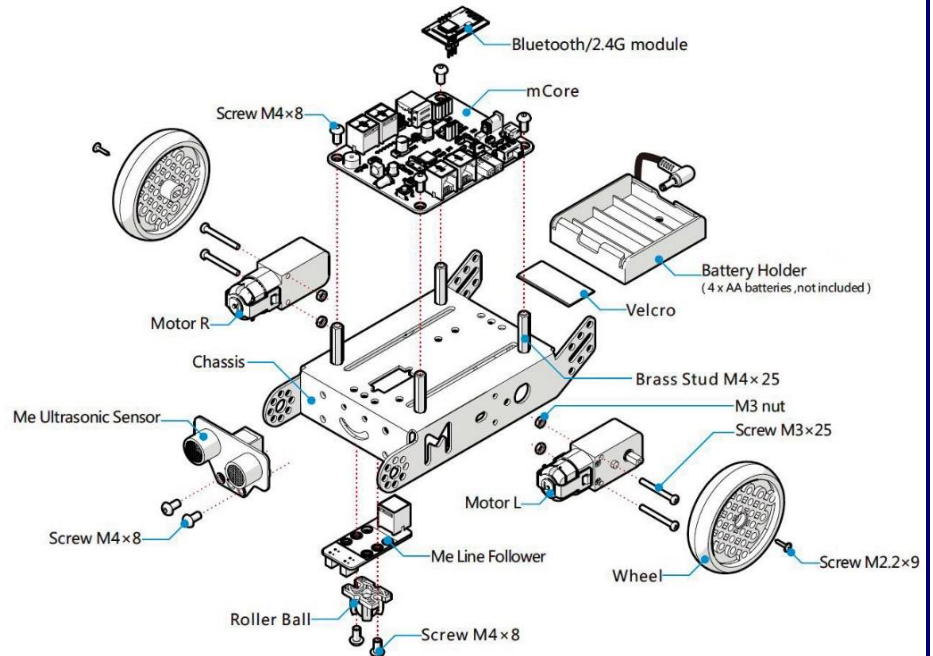
Well, the next time we would meet to do that would be just over a week later - her first day back at school after the Christmas holiday which coincided with the day of the week we normally go to visit. That gave me time to study all about it - '*that'll be a breeze*' I thought!



She had already built Lego models on her own (by following pictorial guides) and setting to work on mBot's out-of-the-box construction, she used her lovely new screw-driver very competently.

I tried to get her to understand about only holding the edges of the circuit boards, but I totally failed in that respect - and in truth, her little fingers were much nimbler when holding bits and putting nuts on to screws in tight places than my big 'Santa' hands.

It took about half-an-hour to complete the construction, but by this time (after a long first-day back at school) she was flagging; but perked-up when she flicked the power-switch and mBot burred for the first-time and came to life. I was relieved that static build-up had not done any damage to its circuits.



We connected it with the serial cable to my Surface tablet and I showed her what I had learned about putting robotics blocks together to make a little programme that did indeed communicate with the robot. Since I had not yet had the chance to buy a battery for the IR remote, we couldn't test that & by tea-time she had had enough.

The following week, we had another go. She was lively to start with but was tiring quickly after another long day. The highlight was the IR remote and the three default modes of mBot operation. She loved steering it around the house and making mBot chase the cat. She thought that line-following on the little figure-of-eight track was brilliant too. I thought to myself however that the task I had been set by my daughter was going to take a long time...

... We are nevertheless making (me in particular) good progress and we have had a few technical hitches so far, but nothing that someone reasonably competent can't work around. These are often genuine robotics issues that robot builders will always encounter at some point. All the problems we have had have been a learning experience (for me) rather than a distraction. As is often the case with computing, you learn more when stuff goes wrong, and you must fix it or find another way around it.

There are lots of robot toys out there that claim to be truly programmable when they really mean they can be controlled from a remote or an app. and mBot is indeed programmable. Many educational (supposedly programmable) robots are quite restricted in what they can do. Whereas mBot is fantastically capable by comparison - and cheaper. With mBot you can design custom robot models that react to sensors, move, play sounds and update lights; and especially if you add the easily available and not too expensive add-on packs like the LED Matrix Display Plate (see Appendix 1, page 184) you can develop useful feedback systems.



Chapter 2 - About Makeblock, Scratch and Arduino

About Makeblock

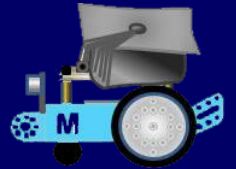
mBot is essentially a children's robot, but it's part of the Makeblock family of beautifully made robotics kits and components for hobbyists (see the image below). You can progress on to Arduino programming with mBot. Teenagers (and Granddads, one hopes!) will be able to move onto other robotics products in the Makeblock range and build projects such as a 3D printer, a robot arm, a drawing machine or a robot of their own design, although much of this stuff *is* expensive.



Makeblock is described as a leading Chinese technology and robotics construction company providing a platform for makers, DIY hobbyists and educators. They raised funds in a 'Kickstarter' crowd-funding campaign about five years ago to establish one of the company's most popular products - the mBot kit. They have developed Arduino based hardware, robotics hardware, and Scratch based software, providing educational tools (using robotics) for learning programming, engineering, and mathematics.

Their robotic kits work with mBlock (a variant of Scratch), a programming language that lets users easily control their robots' movements by using programming software which is made up of pre-loaded colourful and modularized drag-and-drop graphical blocks. Using it, children feel that they can easily programme mBot without writing difficult code or using textual programming language.

Although the company first started out creating robot parts for the do-it-yourself 'Maker' community Makeblock has since shifted its focus to Science, Technology, Engineering and Mathematics 'STEM' educational robots with the belief that it is essential to encourage creativity and innovation in the younger generations.



There are two versions of the mBot, a Bluetooth version for home use and a 2.4 GHz Wi-Fi version designed for classroom use.

Everything that you need to build it is in the neatly packaged box and you need to assemble it using the precise instructions provided and then add your own batteries. There are 45 pieces and it's easy to put them together in about 15 minutes. Once you put everything together, mBot can move around and avoid obstacles on its own, follow a line or just accept commands from the supplied IR remote control.

mBot
One Robot Per Child



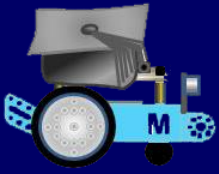
With the Makeblock app for phones and tablets you can control mBot via Bluetooth. For children especially, it can be a surreal experience to actually build something out of tiny parts and see it come to 'life'.

mBot is designed to be tinkered with, and the idea behind Makeblock's mBlock programming interface is that younger children can start out with graphical programming and move on to text-based programming as they become more advanced. This is what they hope will inspire the next generation of engineers. Makeblock Me Series Modules (their plug-and-play electronics circuit board components) are designed to be simple to connect and easy to programme and each Me module comes with its own Arduino library for easy programming.

Makeblock aims to help children build robots and learn to write computer programmes in a fun and educational way as the trend for incorporating coding (the latest educational buzz-word) and robotics into education becomes desirable. The company believes that their kits will help users develop logical thinking and gain the mindset and thought-processes of a programmer - although their systems are not just for aspiring programmers.

About Scratch

Kids from 7 or 8 years old and upwards (inc. their grandparents!) can learn to write simple programmes for mBot using Makeblock's very intuitively simple to use software mBlock 5 (a variant of Scratch 3) and using this free mBlock software is simplicity itself and causes no problems with the majority of computer operating systems.



mBot and Me *a Beginner's Guide*

Scratch is both a programming language and *an online community* where people can programme and share interactive media such as stories, games, and animation with others from all over the world. Scratch is designed and maintained by the Lifelong Kindergarten group at the **MIT** Media Lab.

Massachusetts Institute of Technology Media Lab came into being in 1980 with the aim to invent and then reinvent how humans experience and can be aided by technology. Their current Scratch software is provided free of charge and can be downloaded from:

<https://scratch.mit.edu/download>

While Scratch is primarily designed for 8 to 16-year olds, it is also used by people of all ages, including younger children (with some help from their parents, grandparents or teachers). The current version available to download is Scratch 3.0 and I would recommend downloading it as a learning counterpoint to mBlock 5. It generally has much better help features and has an excellent '*Wiki*' (a website on which users collaboratively modify content and structure). This can be accessed from:

<https://scratch.mit.edu/search/projects?q=wiki>

As children create with Scratch, they learn to think creatively, work collaboratively, and reason systematically. They can programme their own interactive stories, games, and animations by moving small pictures called '*Sprites*' on top of a backdrop called a '*Stage*'.

*Learning **Scratch** is a good and a VERY easy way to understand many programming concepts, but as an ICT teacher I personally achieved many of its goals by teaching kids by using instead the universally used Microsoft Office multi-application environment.*

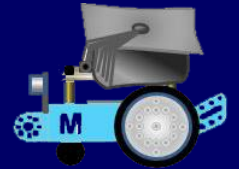
I used to use '*PowerPoint*' to develop logical sequencing skills (this was such a good introduction to programming things) followed by using '*Visual Basic*' (Microsoft's own powerful Macro programming language) which has a slightly different variant inside each component of '*Office*'. My favourite component for this was '*Excel*' where high-quality user interfaces could be developed.

My granddaughter had already done some simple block programming of interactive stories using **Scratch Junior** (designed for kids aged 5-7, using touch-screen tablets) on an iPad at her primary school and she loved it too! The first time we opened mBlock, I was impressed when she showed me how to colour in a background for the stage.

Earlier Scratch and mBlock environments were written in Adobe Systems '*Flash*' and based on open-source software. Importantly, neither Scratch 3 nor mBlock 5 now no longer require Flash to be installed to run. mBlock 5 works very well and Makeblock provide a very helpful support service and an excellent forum where you can interact with other users as well as their own '*Technical Support Department*'.

It's worth pointing out that the Scratch community also provides tons of help with general scripting (but not robotics scripts) problems.

You will find that programming problems can be a learning experience rather than a distraction and, as is often the case with computing, you learn more when stuff goes wrong, and you must fix it or find another way to solve a problem.



About Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino hardware boards are relatively inexpensive compared to other microcontroller platforms and Arduino software is easy-to-use for beginners, yet flexible enough for advanced users. It runs on Mac, Windows, and Linux platforms.

Arduino consists of both a physical programmable circuit board (often referred to as a microcontroller) and a piece of software, or IDE (*'Integrated Development Environment'*) that runs on your computer. In earlier versions of mBlock software, this Arduino programming software was installed automatically on your PC (as a separate entity) when you download mBlock.

In the latest (totally rewritten) version of mBlock (mBlock 5) there is now an Arduino Editor built-in with one-click switching between graphical programming Scratch blocks and *'Arduino'* code. mBlock 5 has a built in editor for programming in *'Python'* code too.

Arduino came to prominence as a tool to help designers, artists, and musicians access the power of inexpensive 8-bit microcontrollers by learning *'a little programming'* without having to acquire the full range of skills needed to work with embedded electronics. mBot has an *'approved'* Arduino derived and enhanced board (**mCore**) at its heart. The Makeblock mBot is highly regarded and although it is fairly expensive in its own right, it is considered to be the most widely available and one of the cheapest robot kits, with hundreds of thousands distributed around the world.

Arduino based boards (remember, mCore is one of these) don't have an operating system per se, so whatever is loaded into the board's flash memory is what will be run at power-on. It is therefore possible to compile code and substitute your own programme for mBot's default code by *'uploading'* it into mBot.

To enable mBot to interactively communicate with the mBlock 5 software, you need (as updates become available) to upload to its mCore board the latest upgrade of special *'firmware'* which interacts with mBlock at low level allowing you to access the Arduino board features interactively from the graphical programming environment. If mBot's flash memory is overwritten by your own customized code, and you want to clear this code, then you will need to do a firmware upgrade, and also upgrade mBlock 5's software with the latest available version - see Chapter 9 (page 37) for more on this.





Chapter 3 - About mBot the Robotics Device

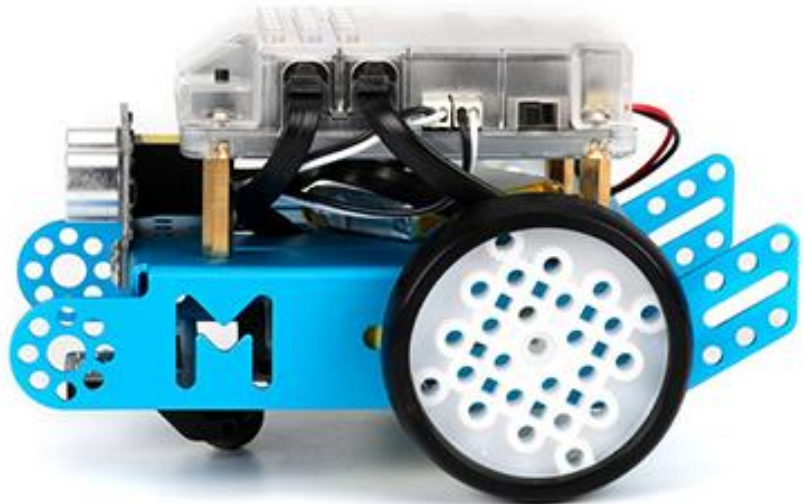
Currently available is the upgraded v1.1 mBot. You can buy mBot robots and related add-on packs online and the cheapest options are through Amazon or via Ebay. Compatible specialist components *are* available too, although many of these are not easily obtainable in the UK.

mBot is very appealing to children and many will get huge satisfaction from its ultrasonic sensor 'eyes' and smiley face and they take to this friendly look straight away.

There are some things that you need to understand to get the best from mBot but Makeblock have done a fantastic job with the packaging, instructions and the whole 'out-of-box' experience.

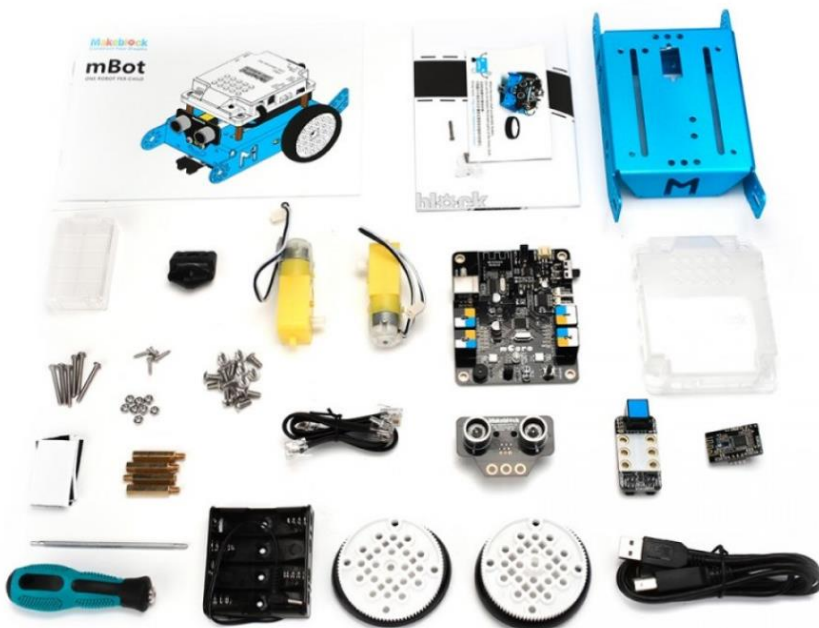
The only tool required to build it (Emma's favourite reversible screwdriver) is included, and the instructions are easy to follow. A typical 7 or 8-year-old *can* make an mBot without much help and it comes in a heavyweight cardboard box which can be kept for storage of the robot and its cables.

Kids get a sense of achievement from building the robot, and it goes together quickly so they won't get impatient. The mBot chassis is made from anodised aluminium - it's beautiful and feels solid and of very high quality.



The sensors and motors connect easily to the board, with no fiddly clips, jumper leads or breadboard to worry about. The board has high-quality RJ25 sockets which receive push-in connectors. These ports are clearly labelled, and colour coded which is important when it comes to programming them later.

N.B. Do hold all circuit boards by the edges when fitting them in place; and after you have opened them do not throw away the little coloured polyethylene bags which contained your electronics circuit boards - you should use them again if you need to store any modules not required in any robot modification project.





mBlock 5 is essentially a visual software development system; but at the outset, only when the mBlock 5 software on your PC and the mCore board on mBot *are connected to each other* by either cable, Bluetooth or wireless can you control your robot.

Changing the colours of the onboard mCore LEDs using mBlock 5 is a simple and fun activity to get started. mBot has a small buzzer capable of playing musical notes too. Since I learned to programme Clive Sinclair's 'Spectrum' in the early 1980's, programming pitch changes for mBots piezo buzzer gave me a sense of déjà vu (and probably will too for everyone old enough to remember it!). The ultrasonic (proximity / distance) sensor can be used for detecting things that come too close. For a novice programmer it's fairly exciting; and experimenting with feedback from it is fairly easy.

Buying add-on components, as soon as you feel able too is a good idea and I would recommend the LED matrix digital display as your first add-on since it opens-up many opportunities for simple but very effective programming by drawing images to display or setting a scrolling message or count-down timers. See Appendix 1 (page 184) for more on this. Emma particularly enjoyed using the '*immediate feedback*' potential of LED panel graphics.

It is reasonably easy to get many of things such as the LED panel working with a bit of help from items posted on the very, very useful MakeBlock forum which can be found at:

<http://forum.makeblock.com/c/makeblock-products/mbot>.

mBot comes with default (Factory Setting) firmware which responds to a small infrared remote-control handset which is included in the box. This means that once you have sorted out power, connections and firmware updates you can immediately play with mBot as a remote-controlled toy. It took about four weeks before we spotted that the IR remote had a removable protective film around the buttons!

The infrared remote is immediately usable from the box to control mBot, but it is also programmable via mBlock 5 which is fantastic since you can create your own key-press instructions.

This means you can control the robot with your own commands via the remote control even when it's not physically connected to a computer. This was a very popular initial programming activity for us, and Emma loved being able (eventually) to write very simple scripts for the IR remote to make mBot do what *she* wanted.

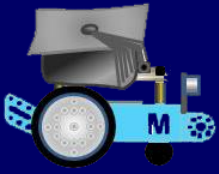
You should be aware that the IR remote control will control *any* mBots in range - this can be fun for coordinating several mBots, but not such fun when two kids want to play with their own mBot in the same room. See Chapter 7 (page 28) for more details on the IR remote.

Electric motors like those that power mBot are often mismatched, very slightly, (which is a common problem in robotics). This can make the IR control steering experience less than perfect, and you can with a little experimentation set one to run marginally slower than the other to achieve a straight track e.g.:

A screenshot of the mBlock 5 software interface. It shows a blue block with a gear icon on the left. The text inside the block reads: "left wheel turns at power 50 %, right wheel at power 49 %". The numbers 50 and 49 are inside white circles.

It is also possible to run the drive motors *individually* in mBlock 5 using the following *extension* block:

A screenshot of the mBlock 5 software interface showing a green extension block. The block contains the text: "DC motor motor port1 clockwise rotates at power 50 %". There are dropdown menus for "port1" and "clockwise", and the number 50 is inside a white circle.



mBot and Me *a Beginner's Guide*

Beware though, full re-programming of the button controls is rather more complex and only works when your scripts are uploaded to the flash memory of mCore, the Arduino board which is mBot's brain.

Despite what I said when mentioning add-on packs earlier, there's much that you can do with the basic mBot so there's not really any need to buy anything extra to start with. There's plenty of experimentation to be had with your own scripts to keep you busy, but it's good to know that you can extend the robot for not much extra outlay when you need something new to try. Being part of the MakeBlock family, it means that there are quite a lot of extra components that can be used to upgrade mBot, enabling you to build new projects. The basic add-on kits are easily resourced in the UK but many other (desirable but expensive) components are not.

Programming mBot will probably inspire you to learn more, enabling you to link robotics with complex programming of sprite output on the stage too.

Many other toys and kits that are available may be brilliant for a few hours or perhaps a few weeks and then kids have done everything there is to do with them, but mBot is not really like that at all. Many young users have been completely inspired to design their own robots and then are inspired enough to be constantly planning and working on their own fantastic ideas

The box that mBot comes in can be opened without destroying it and it is sturdy enough to use to store your assembled mBot.

If you remove the inner packaging then your completed mBot robot fits nicely in the box with space for the IR remote, the cables, spares, tools etc. It might sound trivial, but it's a nicely thought-out and well-designed touch from Makeblock. Parents be thankful - it's always hard to keep track of cables, spares and add-ons for techie toys - but not in this case!

I started this way too, but as you'll see a little later (in Chapter 17, page 156) that I eventually needed to, and did, succumb to bigger and then even bigger storage options.

Basically, and with a few provisos, mBot is indeed a very impressive piece of kit.

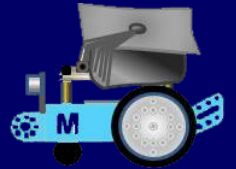
All of Makeblock's products (not just mBot) are of the highest quality and are so well finished and packaged! mBot is a perfectly affordable robot for techie families where an adult can give a bit of help to get children started. mBot is also perfect for schools or robotics clubs with specialist teachers.

Emma whizzed through the basic learning package on her tablet. For any child, writing simple scripts to programme colour-changing lights is fantastic, as is programming mBot to move around with their own commands. Then they've got the light sensor, the ultrasonic sensor and the line-follower to explore; let alone the ability to play (sometimes annoyingly repetitive) musical notes through the on-board buzzer.

Plus, the excitement of programming the infra-red remote themselves.

And that's before you even consider the such good value add-ons such as the aforementioned 8 x 16 LED matrix display panel pack and the Servo add-on pack which can extend what you can do with mBot in so many more ways ...

... keep reading - there's lots to learn and lots to do!



Chapter 4 - About mBlock 5

mBlock 5.0.1 was released by Makeblock in **January 2019** at the same time as the latest iteration of MIT Media Lab's parent application Scratch 3. I did not install any of the Alpha or Beta pre-release versions, preferring to continue my programming journey by using mBlock 3 until the full-blown version of mBlock 5 was made available and stable; and despite using mBlock 3 in quite some depth for much of 2018, I have had to work much harder than I expected to understand exactly how this newly released version works and how it should be used; and *nothing I have read about the application so far has fully explained the concepts of its usage from a beginner users' point of view.*

After about ten weeks of experimenting with mBlock (v.5.0.1) in the early part of the year, what was apparent was that there was little clear, detailed and succinct guidance to this major application update.

In the second part of the year after another version update, nothing has changed. You have to search the internet and indeed Makeblock's own support pages very hard to find anything of value to familiarise yourself with mBlock 5. Even the helpful 'User Guide', accessed from the 'Tutorials' icon on the right of the mBlock 5 title bar (in the form of an HTML file provided by 'GitHub'), does not give you a fully descriptive or clear overview of the *nuances* of how mBlock 5 should be used. Despite some limitations it is nevertheless such a useful reference that I bookmarked this page in my web browser to be able to access it directly without having to open mBlock 5 first.

N.B. You can find the 'GitBook Quick Start Guide and FAQs' using the following internet link:

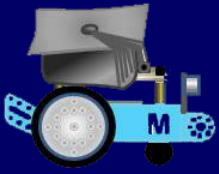
<http://www.mblock.cc/doc/en/>

I do like what I have seen of mBlock 5 so far and despite some minor, but I hope constructive, criticisms noted in this book I am looking forward to using future *improved* releases. I am looking forward too, to the release of more mobile devices variants of the software; since the majority of Android (Intel x86 chip) systems (including my very new Galaxy Note 9 phone) are not yet supported!

mBlock 5 has been specifically designed by Makeblock to support their robotics products and is based on Scratch 3 and Arduino code. Supported robots are 'mBot', 'mBot Ranger' and 'Codey Rocket' together with their 'HaloCode' board and their 'Neuron' app. There is also support for the programming of several Arduino boards including the versatile 'Uno' board and the BBC micro-bit (*micro:bit*) too.

mBlock 5 is available for desktop Windows PCs and mac OS platforms and there is also an mBlock 5 Mobile App which currently runs on Apple iOS 9.0 (and above) devices together with Android 5 and above devices (but currently, Arm-based chips only). Makeblock say that the Mobile App integrates concepts of programming into different and increasingly difficult game levels to keep interest in coding going by unlocking new programming skills step-by-step.

Some of the graphical improvements featured in mBlock 5 (and the equally new Scratch 3) are a great improvement; especially the one which shows a 'shadow' indicating that blocks are within range to connect to each other when connecting and disconnecting them. The shadow feature is again used rather nicely when blocks are dragged back over the categories pane to delete them; and greatly improved too, is the 'illumination' of the user-defined content windows within blocks which indicate that 'nesting' another block inside that window is imminent.



mBot and Me *a Beginner's Guide*

There has been much hype about the virtues of mBlock 5 as a programming tool which is versatile and user-friendly enough to offer users whatever the latest iterations of the parent Scratch application can give. Scratch 3 (developed by MIT) is the latest evolution of Scratch which is one of the most popular computer programming languages for children in the world. It is available in more than 40 languages and users can create, share and mix projects on many different hardware platforms. For help with programming using mBlock 5 there are several (mostly created in mBlock 3) 'how-to' videos available on 'YouTube', and the on-line mBlock community allows users to share projects and learn from like-minded creative individuals.

Makeblock says that mBlock 5 provides both graphical (block-based) and textual programming languages within the software and their aim is to deliver the best in STEM / STEAM education in coding and robotics. Although mBlock 5 is fundamentally for robotics programming, with this software users can drag programming blocks about in the same way as in its parent application (Scratch) to design and create stories, games and animations and all without needing any additionally connected hardware. The raison d'être of mBlock 5 is to enable users to programme hardware devices; the Makeblock family of robots, the BBC micro-bit (micro:bit) and the very popular Arduino Uno board using either drag 'n drop block programming or (with increasing skill levels) textual programming languages.

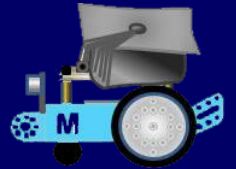
Whilst the novice programmer can experiment fairly comfortably using drag n' drop blocks, they certainly can't try to write Arduino code in the editor and Python whilst being easier and clearer to understand than Arduino is not that simple and mBlock 5, critically, gives no actual guidance on how to use either of these editors (although the '*GitBook Quick Start Guide*' does have a good basic introduction to Python).

The transition for any adventurous user into studying or using 'real' programming code is now fairly seamless; and it is possible to see either 'Python' code as a comparison to any 'Sprites' tab scripts created or 'Arduino C' code as a comparison of 'Devices' tab scripts. In either, the code can be copied to the clipboard and pasted into the respective editor for experimentation.

N.B. Python is an interpreted, high-level, general-purpose language designed to be easily readable, notably using significant whitespace to make its formatting visually uncluttered. The traditional use of semicolons after statements is optional and it frequently uses English keywords where other languages use punctuation. Some Python expressions are similar to those in languages such as C and Java, but it does not use curly brackets to delimit blocks of code.

The new paint and sound editors offered in both Scratch 3 and mBlock 5 make it easier to manipulate characters, music, and sounds. It is now possible in both applications to detect and interact with motion and sound with the video-sensing features of a web-cam; whilst the sound editor has been redesigned so that it is even easier to record sounds and many new sound effects manipulation filters ('faster', 'slower', 'echo' etc.) have been added.

mBlock 5 comes with a cloud storage service too which is specially designed for **A.I.** (*Artificial Intelligence*) and **IoT** (*The Internet of Things*). Makeblock state that these features allow users to master the fundamentals of some of the latest cutting-edge technologies. It also integrates Google's deep learning library of open educational resources for problem-solving using A.I. and '*Cognitive Services*' - an evolving portfolio of machine learning algorithms for building intelligent applications by adding features such as understanding spoken and written language together with facial, speech and human emotions recognition.



Once you come to realise that mBlock 5 has in reality two different Scratch programming sections to use (with mostly different blocks in each of them) it is easy to see where you are going to create '**Device**' scripts that control mBot and '**Sprite**' programming scripts that can be used to animate realistic graphics.

Understanding how '**Broadcast**' messages can be used to transfer data-on-demand between the two to enable you to display sensor feedback from mBot visually on mBlock's '**Stage**' or clicking meaningful sprites to send control commands back to mBot is vital. By using your own graphics to create high-quality interfaces, feedback data from mBot can be represented in a variety of real-time digital, analogue or alphanumeric output formats; and representations of switches etc. can provide real-time controls for mBot too! (See page 75 and Chapter 13 for more on this).

mBlock 5 v 5.1.0 (released in early July 2019) - Modifications and Features

The user interface (the '**Edit**' page) is an improvement on the January (v.5.0.1 release). The '**Scripts**' area is larger and there are now just two choices to adjust the script area size (& therefore the stage size & sprites list size too).

Both the '**Edit**' page and the full-screen '**Presentation-Mode**' display seem to be sharper brighter & clearer. Sadly perhaps, the '**Presentation-Mode**' surround screen is no longer dimmed down (to nearly black) as it was in mBlock 3 and it is now only slightly dimmed; which does have the advantage of seeing a looping script 'pulse' etc. - but that is not really a lot of use and for the most part is actually intrusive since it detracts from the contents on the presentation screen itself. If the surroundings were totally black then this would be so MUCH better.

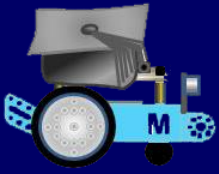
Projects still do not remember which of the three *variable monitor windows* types were last active when they are reloaded in any new software session. They do remember that they were set to show on the stage and where they were positioned, but they always revert back to the default 'labelled' monitor type.

A great improvement though is the much needed addition of an 'Export' sprites facility enabling reuse of the sprite, its block coding and any costumes attached to that sprite in other projects. Attached (or floating) comments remain attached to exported sprites too.

Right-click on a sprite (or the device in the 'Devices' tab) and 'Export' becomes an available choice. This saves a single sprite as a .sprite3 file. These can be added into another project by using the 'Upload' button in the sprites library. A sprite containing block scripts uploaded this way is NOT however added into your 'My Sprites' library. A 'Devices' tab device icon can also be exported, and if uploaded then the resultant .sprite3 file will open as a new device in the 'Devices' tab of a project remembering both device type and any attachments - this seems to be a very quick way to start a new project with some useful device stuff already in place!

It is now possible too to export a .png image of all of the coding blocks written for any sprite by right-clicking on the 'Scripts' area for that sprite (or device) and choosing 'export all scripts to image'.

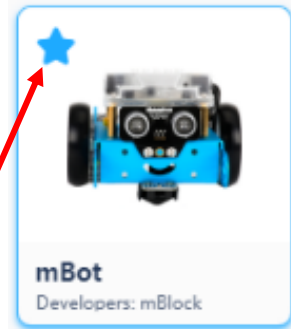
A sad loss from the 'Menu' bar is the button giving instant access to 'My Projects'. To choose a project you now (rather more slowly) have to select 'Open' from the 'File' menu to access the 'My Projects' screen - it is slower, but it's OK though (when you get used to it!).



mBot and Me *a Beginner's Guide*

Usefully, 'Regularly Used Devices' can be set to appear each time you open a 'New' project.

To make mBot your default start-up device, open the 'Device Library'. In the top left corner of all device icons in the 'Device Library' list is a hollow star. Click the hollow star inside the mBot icon and it will turn solid blue.



There is now a supposedly better compatibility between mBlock 5 and Scratch 3 to seamlessly import projects to-and-fro from each other (enabled by just swapping the file extension from *.mblock* to *.sb3*) *but this principle seemed to work well before, so no change detected here!*

Existing extensions have had several updates over the last year and several new extension packs have been added too, namely: *Data Chart, Google Sheet, Translation, Speech, Motion Sensing, User Cloud Broadcast* and *Upload Mode Broadcast*.

However, most of the extension packs (both those for 'Devices' and those for 'Sprites') seem to be of rather limited use! Makeblock have now unlocked the 'Extension Center' too to enable users to create their own extension block sets. I have yet to try this.

In the parent application (MIT Scratch) you are restricted to and constrained by the 'graphical jigsaw' (block-based) options provided and there has in the past been some criticism of Scratch as an introductory programming language due to its drag-and-drop visual style; suggesting that it gives children (its target audience) the wrong idea of programming and it being watered-down compared to other programming languages such as C++, C, Java or JavaScript.

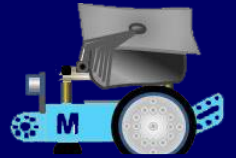
The new mBlock 5 variant of Scratch 3 now addresses this supposed lack of showing 'real' programming language syntax in its parent application by allowing users to easily switch to either the 'Arduino' editor (for 'Devices') or the 'Python' editor (for 'Sprites') with just one mouse-click - see Chapter 5 (page 19) for more on this.

mBlock's much vaunted concept is to enable users to grasp how to programme with blocks first, and then "*effortlessly move on*" to seeing how their programme looks using text based coding. mBlock promotional material makes much of this one-click switching to 'Python' but rather disappointingly, there seems to be no mention that 'Arduino C' is the language required to programme connected robotics devices.

Whilst the novice programmer *can* experiment fairly comfortably using drag n' drop blocks, they certainly can't try to write Arduino code in the editor and Python whilst being easier and clearer to understand than Arduino is not that simple.

mBlock 5, critically, gives no actual guidance on how to use either of these editors (although the 'GitBook Quick Start Guide' does have a good basic introduction to coding using Python to programme sprites).

Makeblock tell me that their company support focus is on their own official software (mBlock 5) and they say that they are not prepared to give much guidance to programming in Python or Arduino, suggesting that users should learn to use these languages by themselves.



Chapter 5 - About the mBlock 5 User Interface

The mBlock 5 interface looks good, and it should do, because it is very much like its parent application, the newly introduced Scratch 3; but with one notable exception - the presentation / output window, the 'Stage' is still positioned on the left of the screen (exactly it was in the its own predecessor, mBlock 3).

The new Scratch 3 application has the stage positioned on the right of the interface; but I guess that the stage remaining on the left in mBlock 5 has the advantages of the interface still allowing any compiled programming language code to be shown in a pull-out window from the right of the screen; just as it always did.

It noticeably takes about four times longer to start-up mBlock 5 than its forerunner, mBlock 3. On my current computer, mBlock 5 loads in about 12 secs as opposed to 3 secs for mBlock 3. On the plus side though, everything inside the new software works *much* quicker and more seamlessly than it did before; and it is indeed a great improvement on its predecessors!

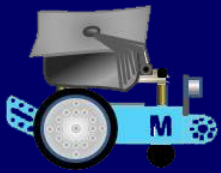
The user interface (the 'Edit Page') now has brighter colours and they are indeed sharper brighter & clearer than they were before. Just compare the title / menu bar screenshots below.

The top image of the two screenshots below is of July's release of v5.1.0 whilst the January release (v 5.0.1) is shown below that. It is obvious from this comparison that the latest version of mBlock 5 is indeed much more vibrant and professional looking, with a colour scheme which echoes the Windows operating system's default colours.



The remainder of the interface is remarkably similar to its updated Scratch 3.0 parent. The newly designed 'Blocks Area' panel of colour-coded category buttons is positioned to the right of the stage and towards the centre (depending on the size of your screen) of the edit page. Next to that is a sub-panel area showing the list of currently selected blocks and on the right of the screen and dynamically expanding to fill the remaining part of the interface is the 'Scripts' (block programming) area.

On the extreme left of the title bar in the January release (v 5.0.1) there was a ≡ 'Main Menu' icon and clicking this gave you the options of creating a new project, opening a project from the computer, accessing help and exiting the application, etc. However, in the bright and shiny July release (v 5.1.0) this has been removed and these important menu items are now grouped under the 'File' menu heading (the *third* icon across the title bar. In my eyes this is a retrograde step and the 'Open' option takes just a little longer to get to the thumbnail based 'My Projects' management page. You create a new blank project page by just clicking the (+) icon here. Sadly, the ☐ icon taking you straight to the same management page of your 'My Projects' files (stored in the Cloud) is no longer part of the new title bar - another retrograde step!



mBot and Me a Beginner's Guide

To the left of the 'File' icon is the 'World' icon and here you can switch to one of thirteen languages for the interface if you need to (a good way to learn some technical French perhaps?). Although mBlock 5 sensibly identifies your computer system's default language when it installs itself.

The new 'Edit' icon to the right of the 'File' icon is rather limited - it just seems to turn 'Turbo-Mode', used to speed-up (fast forward) running scripts for stage presentations either on or off. Next in the title bar at the top of the interface is the default label 'Untitled', this is the 'Title' icon. Click on this to change (rename) the title of your current project.

Immediately to the right of the title icon is the 'Save' ('Floppy Disk') icon. Click here to save a file directly to your 'My Projects' cloud storage area. This is useful for fast saving when working on a project. There are no other save options here so you have to return to the 'File' menu to find the option 'Save to your computer' - using this is important because it enables you can store your files into a purposely structured and hierarchical folder system of your own. Making sequential backup copies of your files to both 'My Projects' in the mBlock cloud AND to your computer is not a bad idea either.

I would advise against clicking the rather prominent 'Publish' icon in the centre of the title bar (for now anyway until you make a specific decision to share your work with others). As soon as you click the icon it will save the project that you currently have open on your screen to the cloud and then open a dialogue (note the screenshot of this below) enabling you to make your work available to the mBlock community.

My projects:



* Project Name

LPR's Interface Project

25

Introduction:

Understanding how 'Broadcast messages' can be used to transfer data-on-demand between the two to enable you to display sensor feedback from mBot visually on mBlock's 'Stage' (in any way that you want) and also enable you to click meaningful sprites on the stage to send control commands back to mBot.

1700

Instructions:

Make it!

1992

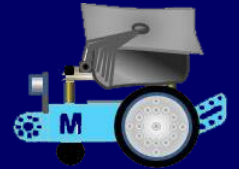
I already read and agree to [Makeblock Community Guidelines](#)

After sharing the project in Community, you can also repost it on WeChat to let more friends see it


Share

Source code

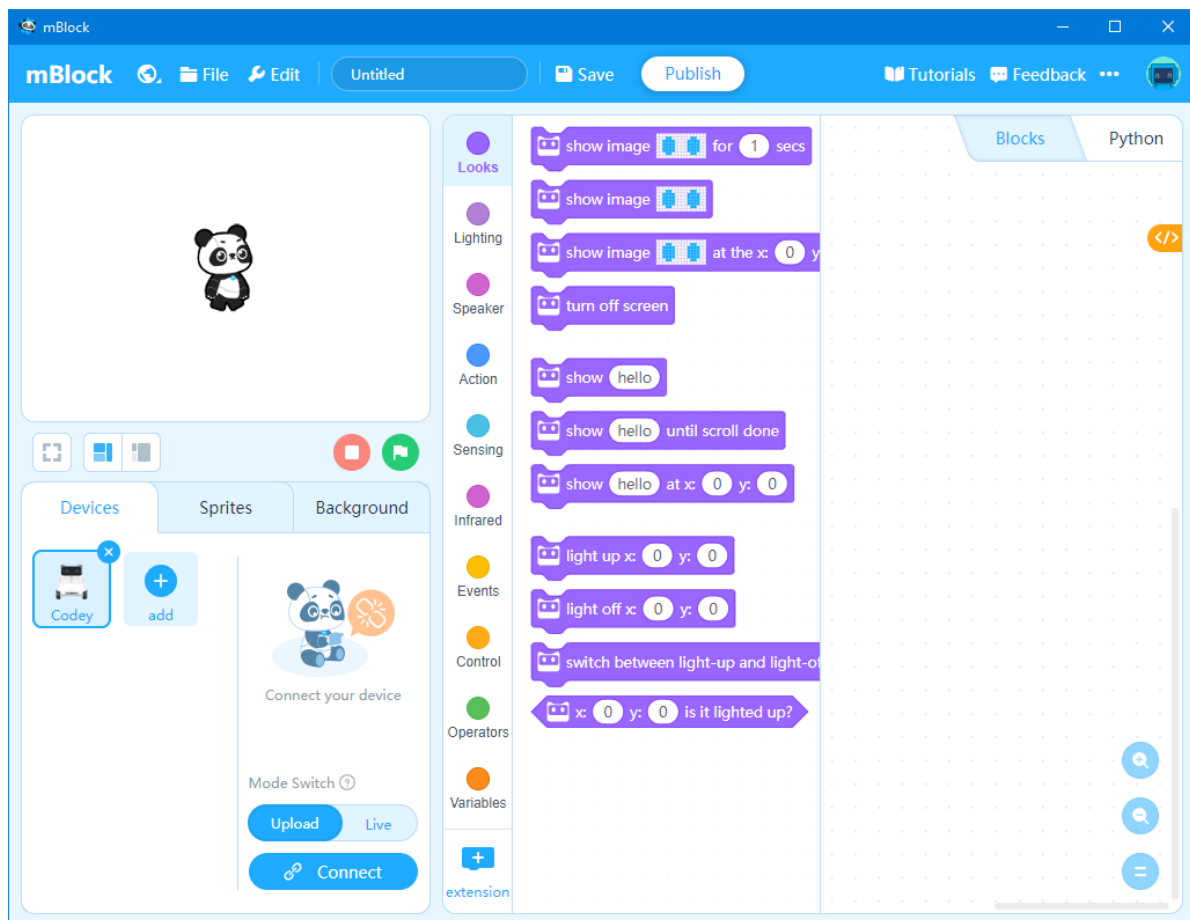
Fortunately, your work is only published and made available to everyone in the community if you click the 'Share' button at the bottom of the dialogue. If you do this then be aware that other community users will now be able to use the file or 'remix' it as a version of their own. Towards the right of the title bar is the 'Tutorials' icon and this is now where the 'Example Programs' management page can be found.



The examples page enables you to access a variety of samples for 'Device' and 'Stage' programming. There have been some improvements to the stuff available, but the example robotics programmes currently available seem to be just for the 'Codey Rocket' robot and the 'HaloCode' single-board computer. Makeblock really should provide sample programmes for mBot users here too.

Use the 'Feedback' icon on the right-hand end of the title bar if you think that you have something important to impart to Makeblock. Response is fast and I think that it is one of Makeblock's great strengths that they have a courteous and knowledgeable team in support of their hardware and software - but you have to be very clear, precise and succinct in what you are trying to tell them. The  icon to the right of the 'Feedback' icon on the title bar allows you to check for updates.

The 'Sign-In' icon at the right-hand end of the title bar looks like it *ought* to contain your picture. mBlock 5 keeps you 'Signed-In' to the mBlock cloud (even if you close down the application itself) until you 'Sign-Out' again. It is now possible to modify some of your account details in your profile and to change your nickname etc. (but not your picture). This icon also gives you access to the mBlock cloud storage service which is a very fast & efficient way of saving / reloading project files; but if you open your project files here it then opens them in *mLink* (the web browser version of mBlock 5). This is rather confusing when you already have the software open and are already using it. The screen-shot image below shows how the mBlock 5 'Edit Page' looks at initial start-up.



The 'Edit Page' is the descriptor for the main mBlock 5 interface and it occupies the whole of the screen below the title bar. It is comprised of three dynamic panels containing six parts or sub-sections.



mBot and Me *a Beginner's Guide*

In the top-left corner is the 'Stage'. This is the core-component display area of the traditional Scratch application and it can be used to display useful visual feedback from 'reporter' blocks and the values of 'Variables' and 'Sensors'. There are three buttons immediately below the stage, the first is the 'Presentation Mode' (full-screen display) icon and then two buttons to adjust the size of the stage. The first of these is the default setting and the second reduces the stage size by about 50% from the default size (and consequently also shrinks the panels below it). The main purpose of shrinking the size of the stage is to increase the working area of the 'Scripts' panel which will now fill most of the right-hand side of the screen.

Also below the below the stage display, are the two buttons to start and stop scripts - the 'Green Flag' button and the red 'Stop' button.

Next, to the 'Stage' area panel and towards the centre of the interface is the 'Blocks' area. Here you can choose from several categories of programming blocks. If you click on any of the colour coded category buttons (see the screenshot on the previous page) then the sub-section to the right changes to show the available programming blocks for that category - this panel defaults to the 'Show' category of blocks.

On the right of the edit page, and initially totally blank, is the biggest panel area (its size dependent upon the size & resolution of your screen display) - this is the 'Scripts' area where you create your mBlock programmes by dragging blocks from the blocks categories and assembling them into script sequences as required.

There are three buttons at the bottom-right of this panel enabling you to zoom-out and zoom-in in 20% increments which enable you to see more or less detail of large block scripts. The (=) button returns the panel to back to its default scale. Any zoom setting that you set is remembered and remains in force the next time you open mBlock 5. I always try to create *short* scripts which fit within the 'Scripts' area without any need to zoom-out or any scrolling up-and-down to see their contents.



At the very top and on the extreme right of the 'Scripts' panel are two tabs. The default is labelled '**Blocks**' - this is where you are right now, ready to create block scripts. The second of these two tabs is there to open one of the two programming editors; so its label will vary depending whether you are on the 'Devices' tab or the 'Sprites' tab. If you are on the 'Devices' tab, then it will be labelled '**Arduino C**' but if you are on the 'Sprites' tab then this will be labelled '**Python**'.

You need to be aware that if you have the 'Devices' tab selected, then clicking on the '**Arduino C**' does exactly that - it opens up an editor for programming in 'Arduino C' not 'Python'. If you switch to 'Upload Mode' and use the (</>) 'pull-me-out' icon (shown on the next page) on the right edge of the interface, this will show any Arduino code generated by your programming scripts.

Very usefully, this programming text can be copied and pasted into the Arduino editor to experiment with this textual programming code if you want to; and similarly, with the 'Sprites' tab selected, then clicking on '**Python**' opens the 'Python' editor.

This too does not initially show any Python code equivalent to your block programming script, but if you click the 'pull-me-out' icon as described above it will also show you the textual code that the your scripts have generated & you can also copy this and paste it into the Python editor screen for further experimentation.



Below the 'Stage' area and the three display buttons is the 'Devices', 'Sprites' and 'Background' categories panel, the contents of which change depending on which of the three tabs is currently active. It defaults to displaying the 'Devices' tab on the left and the 'Connect' / 'Mode' sub-panel on the right.

If the 'Sprites' tab is activated, then the left-hand part becomes the 'Sprites List' and the right-hand panel shows the buttons to access the 'Costumes' and 'Sounds' editors. However, if the 'Background' tab is activated then the left-hand part shows any added backdrops and a (+) button to access the 'Backdrops Library', whilst the right-hand panel remains the same by still showing the buttons to access the 'Costumes' and 'Sounds' editors.

Before you start to control a connected robotics device (e.g. mBot) with mBlock 5 for the first time, a window might pop up asking you to 'Update Firmware' and you should follow this advice by clicking 'Update Now'. It will take 2 to 3 minutes for the device firmware to upgrade and you will need to close & then restart mBlock5 to activate the upgrade.

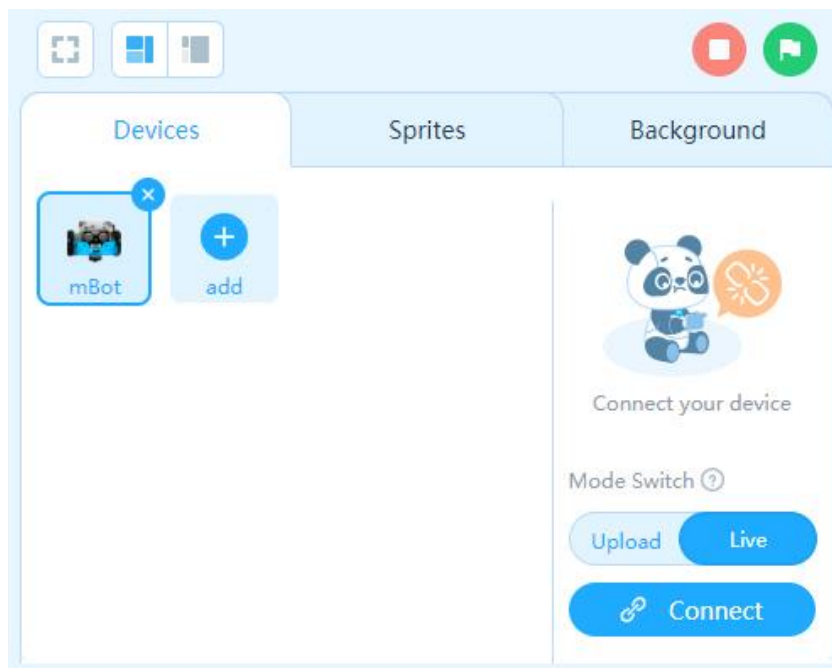
On the right edge of the interface near the top of the 'Scripts' panel is a tab which suggests a fly-out option. When clicked, this opens a sub-window which shows either 'Python' code as a comparison to 'Sprites' block scripts or (if you are in 'Upload' mode) 'Arduino' code as comparison to 'Devices' block scripts.



Both code display types can be rolled back to the right-hand edge of the interface by clicking the close icon which replaces the 'pull-me-out' icon when the code window is activated.



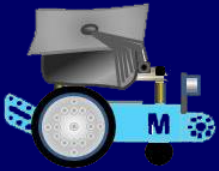
As mentioned on the previous page, the left-hand panel below the stage is divided horizontally, and the bottom half is the 'Devices', 'Sprites' & 'Background' categories panel (one tab for switching to each). But also, part of this sub-panel is taken up with the 'Connect Your Device' area.



This sub-section below the stage on the left of the interface is **very important** and it took me a little time to understand the importance of the first two of the three category tabs that you can select: 'Devices', 'Sprites' & 'Backgrounds'.

So, let's deal with the '**Backgrounds**' tab first. This, in a way, is the least important of these three tabs as it is for adding 'backdrops' or 'scenery' to simple graphics and sound programming activities which use the stage as their primary output (just as if they were projects created in Scratch).

Giving the stage a sensible background rather than its remaining blank does improve the look of the mBlock 5 interface and it is worth noting that a 'thumbnail' of any created backdrop image becomes the icon for your file within the 'My Projects' management page.



mBot and Me a Beginner's Guide

mBlock 5 files saved to your computer do not have the thumbnail mentioned above; but there is a rather nicely designed new mBlock 5 icon instead (shown here on the right).



Secondly, we will consider the rather more important **'Sprites'** tab. *'Sprites'* at their simplest are characters that can be moved about on the stage to play a game or tell a story. Like the *'Stage'* itself, *'Sprites'* are an essential part of Scratch projects where you create scripts for each sprite causing them to move or interact (essentially, to *'act'*!). Sprites can nevertheless become useful graphics to enhance robotics projects - see Chapter 14 (pages 83 to 95). In earlier mBlock iterations, *'Devices'* did not exist, and **you could only add robotics block scripts to *'Sprites'* - but now in mBlock 5 you can only add robotics block scripts to *'Devices'***.

Thirdly and finally; and most important of all for robotics work is the **'Devices'** tab. You can have more than one robotics device available in the interface, but you can only connect to one of these devices at a time.

With the *'Devices'* tab selected, the panel shows the chosen robotics device (or multiple devices) and by default it shows *'Codey'* the *'Codey-Rocket'* robot. There is also an *'add more devices'* (+) button and the *'Connect'* sub-panel is to the right. Here, you simply click the *'Connect'* button to make a connection to your chosen device - but more about the complexities of connecting your computer to a robotics device a little later...

... In the diagram on the previous page I have deleted the *'Codey'* sprite (yes, confusingly, device images are called sprites too!) and I have added the *'mBot'* sprite since that is the device that this book is all about. It is here that you can choose to add another robotics device from the list, e.g. *'mBot'*. To do this, I pressed the (+) icon in the bottom left panel which opened the *'Device Library'* window.

If you double-click on any icon in the *'Device Library'* then it will be added into the bottom panel of the interface (and you *can* have more than one mBot icon here, but I'm fairly sure that this is not really a good idea!). If you do not want a device to remain in the interface (e.g. *'Codey-Rocket'*) then you just click on the (X) button on the corner of the device sprite to remove it.

Device sprite icons in the device library have three states. Newly added and previously unused device sprites have a green plus-sign (+) button in the corner indicating *'new'*. The basic sprite for each device otherwise, is the **'ready to go'** (*OK to use*) icon shown top-right; whilst the sprite **'update needed'** (*this device has a firmware update available*) has a green download arrow button in the corner of the sprite as shown bottom-right.



Click this green button to download the update; but do remember that **you must then close down and restart the app. to enable the update.**

It's very important to realise that if you click on the *'mBot'* sprite here in the *'Devices'* panel it will cause the programming blocks in the *'Blocks'* area to dynamically change to *'mBot' specific programming blocks* from what were by default *'Codey' specific* programming blocks. The number of block groups available to each device changes too; clever stuff, and very fast!

Ignoring the *'Scripts'* panel occupying most of your screen, the left-hand part of the *'Edit Page'* interface looks essentially like the screenshot shown on the next page.



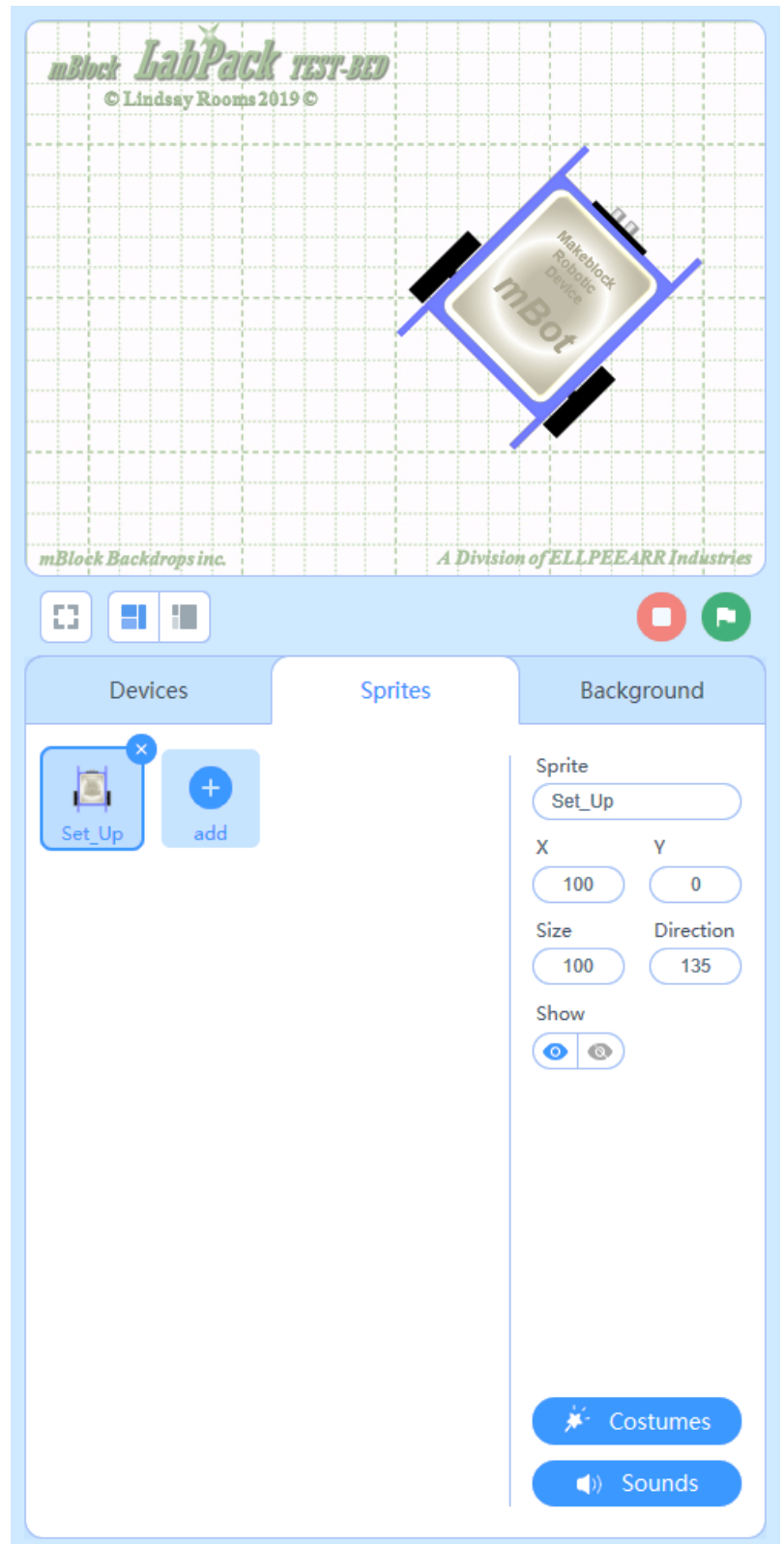
The stage is by default blank, white in colour and shows a 'Panda' sprite in the centre, but I always start I start mBlock 5 with my own "*mBot Setup Page*". 'Set-Up' file from a shortcut on my desktop. This file has the specific backdrop and single sprite icon shown here - but more about this file later.

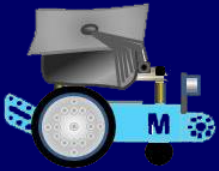
The 'Sprites' and 'Backgrounds' sub-panels shows icons representing any sprites or backgrounds added to a project and these sub-panels have identical 'Costumes' and 'Sounds' buttons. Each button opening the applicable editor for each. The costumes button opens the graphics editor which allows you to edit or create either sprites or backgrounds (depending upon which sub-panel tab you were on when you clicked the button). Use the (X) button to close each editor.

There seems to be a problem with *saving* any graphic that you draw in either editor - such graphics are OK and are useable when a project is created but currently, there is to be no way of adding a newly drawn sprite 'Costume' to the 'Costumes Library' or a background to the 'Backdrops Library'.

Any sprites or backgrounds created elsewhere (I experimented by uploading into the costumes library a file which I had drawn and saved in mBlock 3 last year) can be uploaded and used on demand - but **newly drawn graphics can't be saved as currently there is no obvious save to library button** in mBlock 5.

I tried uploading a test backdrop into the 'My Backdrops' library using one from my old mBlock 3 computer files; but mBlock 5 now limits these to a max. upload file size of 2Mb and I had to adjust it in a graphics editor to get it to upload!





Chapter 6 - Setting-up and Connecting mBot

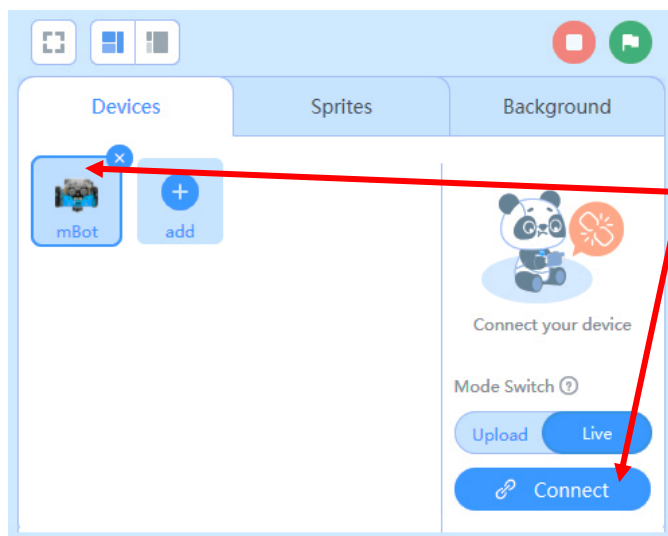
Because mBlock 5 is now written in HTML5 and JavaScript you don't have to download the application to your computer; it can be used 'online'. Just connect your device (mBot) first and then start **mLink** and then open **mBlock Web** using this address:

<https://ide.makeblock.com/#/>

mBlock Web does work very well but installing mBlock 5 software on to your PC to work 'offline' is a good idea. I have already mentioned that the 'Sign-In' icon on the menu bar of mBlock 5 will open **mLink**. You can download **mLink** if you need to from the address shown below; and as mentioned on page 6 you can also download mBlock5 and its predecessor mBlock 3 for a variety of different computer platforms from the following address:

<http://www.mblock.cc/mblock-software/>

Connecting your chosen Device (mBot)



The simplest way to connect your device to your PC is by using the supplied USB serial lead. The first time that I tried this it was quick & faultless; **but you do need to remember to press the 'Connect' button in the 'Devices' panel after the interface opens and after you have selected mBot as your chosen device.**

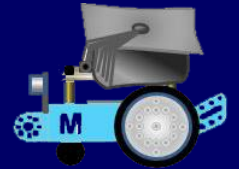
MOST IMPORTANTLY: You must remember to press 'Connect' every time you load in a previously created project from your files.

If the 'Mode' switch (which is just above the 'Connect' button in the 'Devices' panel) is pushed to the right and is blue in colour then it is in the 'Live' (ON) position; if you slide it to the left, it turns grey to indicate the 'Upload' (OFF) position.

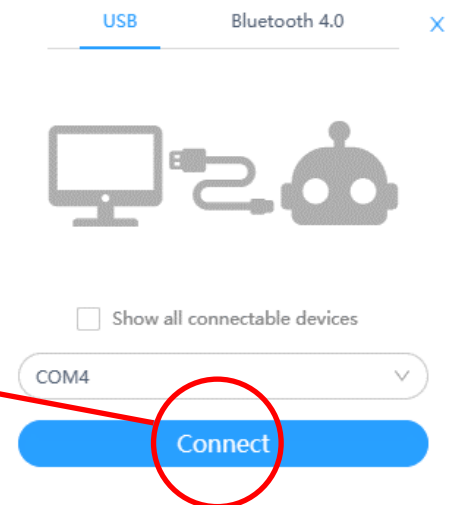
You only need to use 'Upload' mode if you want to upload a script that you have written into mBot's flash memory to enable mBot to run independently from your computer in what is also called 'Offline Mode'. You also need to use 'Upload' mode if you want to see your scripts in Arduino code.

It's opposite, 'Live' is the norm; also known as 'Online Mode' or 'Test Mode' this (for normal, everyday programming usage where you don't have to upload codes into a stand-alone mBot) is where you can control mBot as long as it is connected to your PC by either cable, wireless or Bluetooth dongle.

It doesn't seem to matter if your device; 'mBot' in our case, is switched on and the serial lead connected before you attempt to make the connection - any permutation or any order in doing this seems to work and for the majority of users, this should always work seamlessly.



The 'Connect' dialogue screen (on its default USB tab - shown here on the right) recognises that a device has been connected and will show the COM (serial) port trying to connect. If not, it just waits until you switch mBot on and connect the serial lead between the device and the computer; and once a connection is detected the dialogue box will show the connection port number.



In either case, you then just click the 'Connect' button at the bottom of the dialogue and mBot will beep once to acknowledge that a connection has been made.

As already described, you may find that an 'Update Firmware' dialogue will appear when you connect mBot to mBlock for the first time. This suggests that your device needs an update to its firmware, so just click to accept the update (which is fairly quick to complete).

Remember to close the app. afterwards and then restart it again to enable the update.

Despite Makeblock suggesting that connecting devices to mBlock 5 is easy (*which by-and-large it is*) **everything is not always as straightforward as suggested** and from what I have seen on internet sites, other users have experienced similar connection issues too and, in my opinion, a little more explanation / discussion on the connection of Makeblock's robotics devices would be gratefully received by many users if Makeblock could detail these clearly.

You may have some of these problems too!

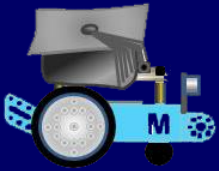
The second time I tried to connect mBot to mBlock 5 it just would not show a COM port to enable a connection to be made - nothing! Later I tried again - and making a connection this time worked OK. This went on and on for days and days with what appeared to be an intermittent connection problem - how strange & why?

I tried what other users had been recommended on the Makeblock Forum - reinstalling the software - no different - running it as an administrator - no different. I also tried rather more advanced techniques like uninstalling all of the USB Hubs in my PC's 'Device Manager' and rebooting to let it reinstall drivers etc. and nothing; just still the same intermittent connection issues.

In my PC's 'Device Manager' > 'USB Hub Properties' > 'Power Management' section I prevented all of the USB Hubs turning off to save power - still nothing.

I manually reinstalled the USB-SERIAL CH340 driver which provides the serial link to mBot too and still had the same (and increasingly annoying) intermittent connection problem. Strangely, the connection between my PC and mBot always worked when I switched back to using mBlock 3 and when I got around to connecting my Surface tablet to mBot using mBlock 5 it always worked there too with no connection problems at all.

So, the problem looked to be with my desktop PC - a hardware conflict somehow, and then only sometimes...



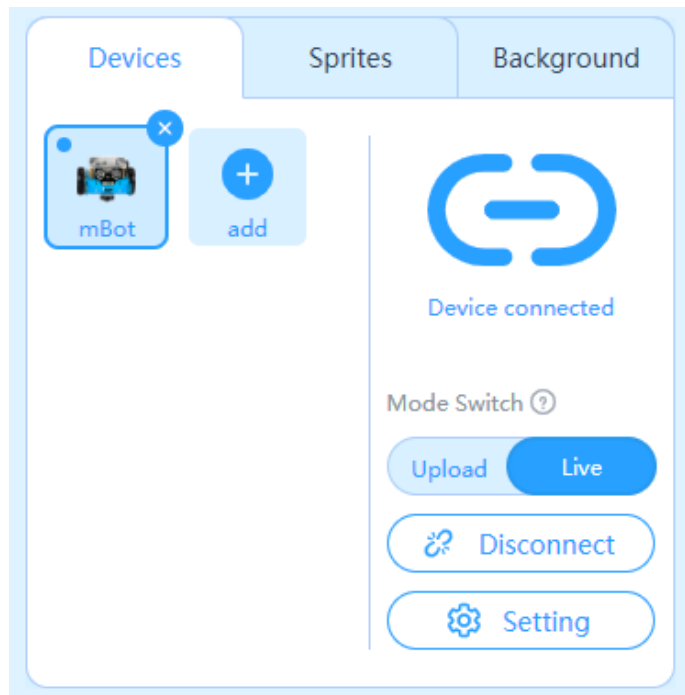
mBot and Me *a Beginner's Guide*

...One afternoon, after a particularly annoying and frustrating session of trying to connect mBot to my PC, I gave up and was on my way out of my study when my printer 'gonged' as it shut down (which it does after a couple of hours of inactivity) a light-bulb moment - THE PRINTER had been ON! So, I went back and restarted mBlock 5 and it instantly saw that the 'Connection' panel showed a connection on COM5. I clicked to make the connection and *mBot was immediately controllable from the PC.*

It looked like the conflict might indeed be with the printer, so I closed mBlock 5 and restarted the printer. Then I restarted mBlock 5 and activated mBot. This time, *no connection* showed in the panel and mBlock 5 was sitting patiently waiting for a connection. Leaving mBlock 5 in that waiting state I turned my printer off - after a few seconds COM5 magically showed in mBlock 5's 'Connect' dialogue box.

YES - it was the printer (which was connected via a USB port on my PC) that was causing the conflict; even though the printer was connected to Printer Port (LPT1).

When I researched this later, it is apparently a known fact that printer drivers often take control of all USB ports (& mBlock 5 didn't like that, although mBlock 3 seemed to tolerate it).



So, a good habit: - Always shut down a printer before trying to connect with mBot.

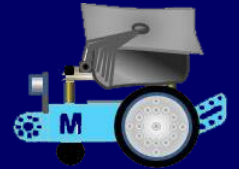
When a connection is made, the connect / connected panel shows 'Device connected', and you can see from the diagram above that mBot is the active device. As well as the 'Upload / Live Mode' switch there are now two new option buttons at the bottom of the panel; a 'Disconnect' button and a 'Settings' button.

Makeblock has not made the significance of these buttons very clear at all. If you click the 'Setting' button, it clears the panel and shows at the top just one uninspiring blue text option entitled 'Update Firmware' - this text doesn't look like a button, *but it is* and if you hover over it then does look like a blue button - click it and a new dialogue box will appear in the middle of the screen and this dialogue box is **VERY** important.

The dialogue box that has just opened shows that the chosen device is mBot and it gives you two choices...

... If you click in the 'Firmware Version' box (see the diagram below) then you have the option to choose to update either '**Online firmware**' OR '**Factory firmware**'.

Online firmware updates mBot with the latest operating system firmware from Makeblock; whilst the other setting, Factory firmware resets mBot's flash memory to the latest update of the default settings.



You may have probably (as recommended earlier) updated the 'Online firmware' to the latest version, but if not, you should choose that option now by pressing the blue 'Updates' button to update mBot with the latest firmware version available.

Device Firmware Updates
✕

Device:

mBot

Firmware Version:

Online firmware(06.01.107) ▼

Cancel

Updates

On completion, the dialogue box suggests that you shut down the device (mBot) & restart it and connect to mBlock again and you should do as the dialogue requests. mBlock will then connect to mBot just as it did before with everything working as it should.

You only need to choose to update 'Factory firmware' if you have for some reason changed the firmware in mBots flash memory by uploading into it (*by design or accident!*) a programme of your own.

N.B. You may also need to update both online and factory firmware as a recovery measure if for some reason mBot goes 'loopy' and start running amok when running a test script.

I don't know why mBot misbehaves but this sometimes happens, and other users have documented it frequently on Makeblock forum pages. So, if you are working with the robot on a table, bench or desktop then beware!

Look upon 'Device Firmware Updates' as a sort of 'blood transfusion' for mBot's mCore brain. Do not be afraid of giving mBot's firmware another 'shot' if you feel that you need to. Other mBot users (particularly in mBlock 3) have recommended device updates as a cure-all panacea and you will find that it usually stops any unpredictable behaviour if you re-run the same script that you were testing before.

Downloading and installing the latest *Windows Arduino Board Driver* from Makeblock is not a bad idea either as part of this global 'blood transfusion'. Makeblock boards use an FTDI chip for their serial connections. In the older mBlock 3 software, there is an option under the 'Connect' menu that says 'Install Arduino Driver' which sets up the serial driver that needs to be installed for Makeblock boards. In mBlock 5 however there is no such option.

I got this driver from Makeblock using the following link:

http://download.makeblock.com/Makeblock_Windows%20arduinodriver.zip

So, there is a lot more to making a successful connection between mBot and mBlock 5 via USB than users are led to believe; - **but what about connection via Bluetooth.**

If you have Bluetooth available on your own PC, then when the 'Connect Device' window pops up **you would think that you are just meant to choose the 'Bluetooth 4.0' tab on the 'Connect' panel** and the Bluetooth port of your device would be automatically detected and you would then expect to just click the 'Connect' button again...



... HOWEVER, mBlock 5 doesn't currently support *any* Bluetooth connection with PC's with **either built-in (or external) third-party Bluetooth!** I have asked Makeblock and they say that they are trying to make mBlock 5 support third-party Bluetooth just like mBlock 3, but to solve this problem it was suggested by their support team that I *had* to buy the **Makeblock Bluetooth Dongle** which has been specifically created to communicate with their robots.

They suggest that the official Bluetooth Dongle guarantees compatibility and they say that they are not sure whether any other third-party Bluetooth dongles are compatible with their robots as *connection issues when trying to use these are individually hard to solve.*

Makeblock say that you just plug their Bluetooth dongle into any available USB port on your PC, power-on mBot, pair them, and start programming.

The Makeblock supplied Bluetooth dongle that I bought for £15 via Amazon in the UK *did* connect mBot to mBlock 5 - **but once again, making that connection was not quite as straightforward as they suggest.**

Theoretically, you just plug the dongle into any available port on your PC and it then very straightforwardly pairs up with mBot which allows them to successfully establish a connection with each other.

But for me, connecting mBot to the mBlock 5 interface was once again a little more complex.



Since this *is* called a 'Bluetooth' dongle, I thought naively (& why wouldn't I?) that you had to choose the **Bluetooth 4.0** tab in the '**Connect**' dialogue window; and try as I might I couldn't get the mBlock 5 software to see any available devices.

Eventually (several hours later) after reinstalling the software, trying different ports, restarting the computer etc. etc. I switched back to the USB connection tab in the connect device window **and just happened to notice that the COM port had changed** from the one I had been using for a serial lead connection (COM4) to the port on the front of my PC's case (COM5) - the port where I had plugged in the dongle.

I pressed the 'Connect' button and 'hey-presto' a connection was made, mBot beeping once in acknowledgement and working perfectly and untethered from the PC for the first time using mBlock 5.

On checking my PC, it showed in the '*Device Manager > Ports (COM & LPT)*' section: 'USB-SERIAL CH340 (COM5)' as opposed to seeing 'USB-SERIAL CH340 (COM4)' which it does when I connect using a serial lead (and the dongle worked in exactly the same way on my Surface tablet too - which has built-in Bluetooth).

The dongle also worked fine when I tested it with mBlock 3, connecting as a serial device (as described above); and it also worked with no conflicts whilst my old external Bluetooth USB Adaptor device was still plugged in the same port on my desktop PC where it had always been.



It's a pity that Makeblock had not CLEARLY mentioned anywhere that mBlock 5 treats the Makeblock BLUETOOTH Dongle as a Serial device not as a Bluetooth device.

The 'GitBook' help file only says: "Makeblock Bluetooth Adapter is a special gadget for Makeblock's products to connect to your computer via the USB without the use of a USB cable. Please refer to the product guide for detailed guide."

'Via USB without a cable' is a clue, but not a very clear one. The guide then goes into to a massive section on Bluetooth connections! The tiny booklet of info. pages that came with the dongle (shown actual size here on the right) *did* say to "Choose the serial port ..." - but in 7-point font, such an easily to miss line!).

I am assuming, *although it hasn't been stated as such*, that the dongle *does* transmit to (and receive from) devices using Bluetooth & it does something actually rather clever in passing this data to-and-fro via the USB port. - If this is the case, then why not say so; clearly!

Do remember that only when the mBlock 5 software on your PC and the mCore board on mBot are connected to each other, can you control mBot by writing 'Scripts'.

I use the USB Serial Cable most of the time and switch to Bluetooth only when I want to operate mBot running free on the ground - a cable does restrict the travel distance of mBot considerably.

The controlling connection can be by either Serial Cable, Bluetooth or, if applicable (because you bought the education version), 2.4GHz wireless. You can solve the problem of communicating with mBot without a wire being attached to it. by using the aforementioned USB (Bluetooth) dongle or the alternative to the standard Bluetooth module on mBot, the 2.4G wireless module. Setting up a 2.4G connection if you have this, is very straight forward. Just insert the 2.4G adapter (receiver) into your PC and pair it with the 2.4G module on mBot and after successful pairing, just click 'Connect'. Do remember too, to turn mBot *ON* with the little switch on the side of the mCore board!

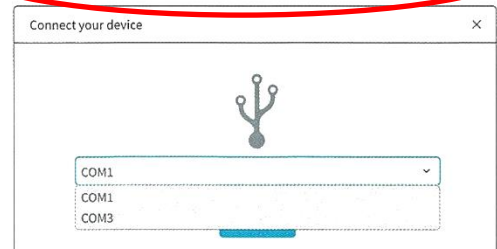
On power-up, and again when the USB serial lead is inserted, mBot outputs three rising notes on its buzzer followed by both of the LEDs on its *mCore* board flashing Red, Green, Blue and then turning White to signify ready. Sadly, the White LED lights remain on and you have to (but don't need to) turn them off using a suitable programming block by setting all of the colour settings to 0. When connecting mBot to your computer the PC produces a short burble of sound in acknowledgement of the connection being made.

The connection made, everything is ready - you are now able to begin your mBot / mBlock programming journey and test your mBot for real.

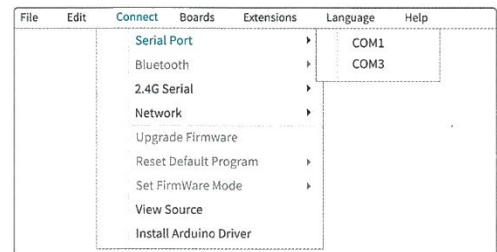
Software connection

Check your current version of mBlock, and begin to connect software according to the following instructions. For downloading and using mBlock, please check: <http://learn.makeblock.com/bluetooth-dongle/>

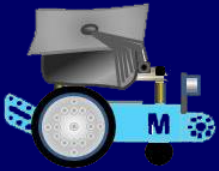
mBlock 5 Choose the serial port for the bluetooth dongle, and begin to connect the device.



mBlock 3 Choose the serial port for the bluetooth dongle, and begin to connect the device.



Note: software update may result in differences between the pictured instructions and the real interface; please refer to the real interface.



Chapter 7 - About mBot's Remote Control Modes

Until you get used to your mBot, it's not a bad idea (after first power-up) to pick up mBot and try pressing the on-board button to see if it switches between its three pre-set modes (**IR control**, **Obstacle Avoidance** and **Line-Following**) - it should, but if that doesn't work then you will need to use the 'Reset Default Program' option described in some detail earlier.

After power-up and connection I usually test that mBot is connected to mBlock 5 by carrying out the following quick test. In the 'Blocks Area' switch to the 'Show' categories of blocks. The topmost block in this category is the stack block shown on the right:



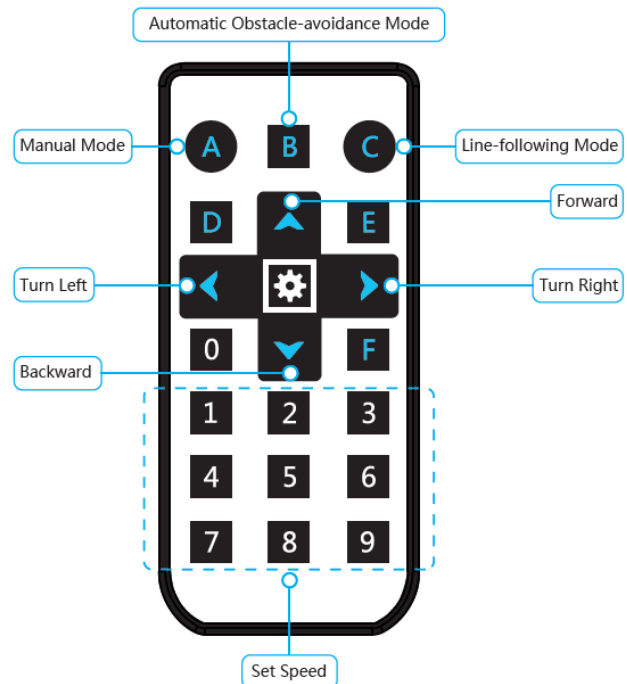
There is no need to drag this programming block on to the 'Scripts Area' of the mBlock interface - but you *could* if you want to. This block sets the colour of the mCore LED lights and if you just click it (see the guidance advice below) then mBots LEDs will both briefly turn red and then go off again.

N.B. A good technique to develop when clicking on programming blocks (to either drag them or activate them) is to **always avoid any areas on the block expecting input** e.g. avoid the 'all' arrow or the 'shows colour' or the 'red' coloured bubble window or the '1' secs bubble window (all of which can be seen in the block above).

About the IR Remote and the 3 pre-set Modes of Default Operation

You can also use the excellent little **IR remote control** that comes as part of your mBot kit to start controlling mBot *without* connecting to mBot at all. mBot has a default factory setting of three pre-set modes, **IR control mode**, **Obstacle Avoidance mode** and **Line-Following mode** all of which can be operated using the IR remote control or, as mentioned above, by pressing the on-board button instead to toggle (step) through each of these options in turn.

Once you have gained some confidence and have started messing about with mBot, you may find that the IR remote control will not work anymore, since the out-of-the-box default functions are no longer loaded. This happened to us very early on and Emma was very disappointed until I worked out what I had messed up and what to do to fix it. I had somehow uploaded a programme into mBots memory by mistake.



Tips: If the speed you set is too low, mBot may not move. In this case, just set higher speed.



When you first turn on mBot, it *should* be in IR control mode by default and the on-board RGB LEDs are both white.

If mBot is in 'Obstacle Avoidance' mode, then the on-board RGB LEDs are green. Place mBot on the floor, and watch it avoid obstacles.

If mBot is in 'Line-Following' mode, then the on-board RGB LEDs are blue. Place mBot on the black line-follower map and watch it track around the figure-of-eight line.

Pressing button A on the IR remote will stop mBot in Obstacle Avoidance and Line-Following modes using, whilst the arrow keys can be used for manual driving /steering and the number keys can be used to set mBot's speed - 0 = slow & 9 = fast!

Remember, you may need to change mBot back to its default 'Factory Firmware' setting if these IR remote-control mode settings do not work correctly. This can (as mentioned on the previous page) be a source of much confusion if you forget to do so.

Controlling mBot using *free* Apps (for Tablets and Phones)



The original **mBot app** was probably the first mBot app to be written by 'Xeecos', an Android developer specialising in robotics stuff. It is no longer available in the Google Play Store.

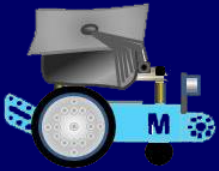
It therefore took me a bit of trouble to find and install this on my old Galaxy SII phone and only seems to be available by APK download.

The mBot app's icon looks like the image shown on the right and its interface looks good; but in truth it really is rather primitive. It is a very basic controller & has essentially the same three default modes as if you were using the IR remote control. I was impressed however that it actually worked on my old SII phone (see the paragraph about my phone on the next page).



This app made a Bluetooth connection with mBot easily and it started online in 'Manual Mode' by default. This is where you can drive mBot with a 'joystick' type controller which has an illuminated speed indicator - this seemed to be programmed incorrectly - mBot turns *left* when the joystick is pushed right & turns *right* when it is pushed left. Forwards & Backwards are OK - funny, but the joystick on the rather similar 'Drive' screen of the much more workmanlike Makeblock app (described on the next page) has just the same error.

The interface has four buttons on the right of the screen - one to toot the buzzer which is very boring and just toggles through 7 pre-defined tones; a 'Speed Up' button which does not as you would expect alter the speed of mBots motors - it just gives a 5 sec. burst ahead with an on-screen countdown timer; and a light button which toggles through 6 simple (& also rather boring) changes to mBots LED lights.



mBot and Me *a Beginner's Guide*

On the left of the interface is a vertical strip containing five more main-function controls:

- ‘Unmanned’ which should be ‘obstacle avoidance’ mode, but it doesn’t seem to avoid things at all!
- ‘Manual’ (as described on the previous page). This is the default control - manual driving mode.
- ‘Track’ which is the standard line following mode, mBot runs slowly, but it works very well.
- ‘G-Sensor’ is possibly the best feature here and uses phone or tablet tilt to control steering.
- ‘Shake’ mode has a switch to set forwards or backwards movement and a then a violent shake of your phone or tablet will move mBot a bit (about 60mm a time).

If you do have a suitable phone or tablet, then you *do* need to install the much more sophisticated **Makeblock App**, also written by Xeecons which can also be downloaded from Google Play store.

N.B. I couldn’t do this when we first got mBot because at the time of testing, my phone was a ‘steam-driven’ Galaxy SII and is not modern enough. I updated the operating system to Android ‘Jellybean’ 4.12 (the highest update possible) but the Makeblock App needs ‘Jellybean’ 4.3 or newer to be compatible. I couldn’t do this with my own tablet either because it’s a Microsoft ‘Surface’ which is not supported. So, to test it I had to ‘borrow’ Emma’s Galaxy tablet whilst she was asleep!

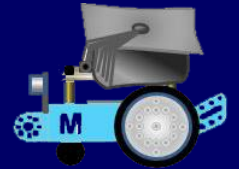
Makeblock 3.4.0 is available for Android (Intel x86 chip) devices and at last I now have it on my new Galaxy Note 9 phone. The Makeblock App is a free and easily downloadable app for both Android and Apple devices that has been specifically designed for controlling a variety of Makeblock robots.



The Makeblock App’s icon looks like the image on the left. It pairs very easy with either a phone or a tablet and it connects automatically with mBot via Bluetooth. Once the Makeblock App is installed, you can play with mBot immediately and it has controllers for all the default functions that you can choose with the IR remote or the on-board button.

On a large screen tablet (or my new ‘phablet’) this is great, but it is quite hard work using this app on the screen of a small phone. The software uses ‘Tiles’ which you scroll through and click to choose options and sub-options and there are the following tiles to choose from initially:

- ‘Play’ - this tile has a choice of 4 modes: ‘Drive’, ‘Draw & Run’, ‘Musician’ or ‘Voice Control’.
- ‘Code’ - this links you straight into the **mBlock ‘Blockly’ App** which Makeblock have developed to teach the basics of coding using mBot & mBot Ranger (described in some detail on the next page).
- ‘Create’ - this tile is probably the best of these options and it is where you can drag controls to build sophisticated consoles (as shown in the diagram on the next page) and then save them. I was amazed to find that the app has controllers for the most popular of mBots optional add-on packs too! There are in fact more than 30 predefined modular components which can control different sensors or execute various commands.
- ‘Build’ - this shows a wonderful (but rather pointless since you’ve probably built it already) 3D animation of how to assemble mBot.
- ‘Expand’ - this opens a sub-menu of several control options for the optional add-on pack models.

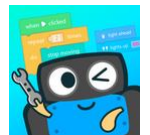


In the 'Create' section, there is a fantastic design mode (just click the 'Design' button at the top to edit your own interface screen & then the 'Play' button to see what your added control icon does). To create your own customized control panels, you just drag different control and feedback devices from a palette to the control panel area and then you just tap on them to edit them. This 'Create' component is a very well thought out, intuitive to use and is a brilliant way for children to interact with mBot.



You can immediately use a joystick-like controller function to drive mBot, changing speed and direction easily. Emma loved it, took over straight away and within minutes had found out how and had effortlessly created and saved her own controller screen. You can choose which ports to monitor and display feedback. These can be in the form of analogue graphs or awesomely good-looking numeric displays (see above). Most importantly, you can execute commands using simple block coding - drag-and-drop style. This is a very good precursor to programming using the excellent features of the 'Code' component (see the next page).

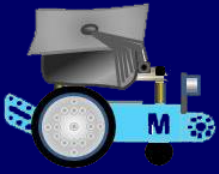
The **mBlock Blockly App** is a *supposedly* 'game-based' app (built once again by Xeecons) for Makeblock programmable robots and it has an icon that looks like the image on the right. It is a simple programming learning centre using drag 'n drop block programming where you can only progress by completing sections in sequence. Makeblock say that it introduces students to the world of robotic programming on mobile devices. They describe their variant of Blockly as 'immersive game-based learning' where you complete tasks and learn coding skills gradually using intuitive block-based programming. They describe these tasks as 'games', but they aren't really games (unless the whole experience of programming mBot is seen as a game!).



These simple tutorials are quite easy to follow but you can't progress to the next level unless you get the current exercise correct and this can be a little off-putting at times if you don't quite get the point of what is being asked (since they are not always described in grammatically correct English).

Also, whilst you are in the tutorial there is no opportunity to experiment with scripts of your own which is also a bit frustrating for a bright young user since each exercise only makes the relevant blocks required for the exercise available, leaving most of the block categories greyed-out; and moving on to the next exercise is the only option! I was amazed how fast Emma worked her way through each of the exercises (although she did get stuck once) and she was very pleased with herself when she completed all of the tasks on each of the ten levels.

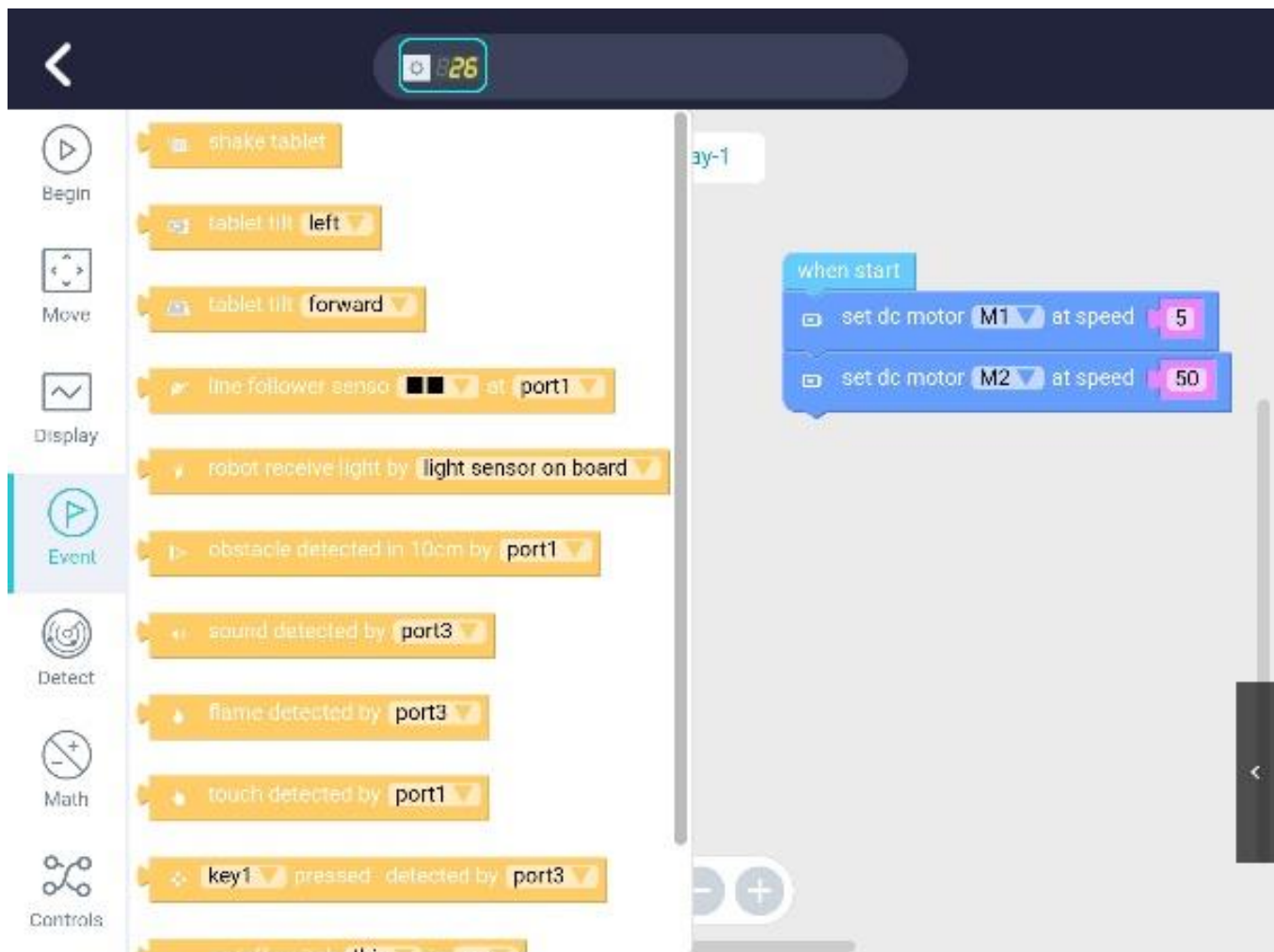
However, this app. does have an annoyingly incessant background track sound (and 'Mr Panda' as a fairly sedentary 'instructor') and it is also hard to exit since there is no back button allowing an easy return to the main tile (menu) group.



mBot and Me *a Beginner's Guide*

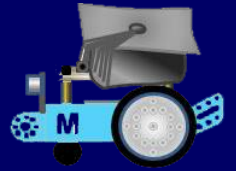
The tutorials though are fun nevertheless and I think that I would recommend this as a first step to programming mBot for young children since they do get the satisfaction of making the robot do something themselves at every stage of the ten levels of instruction. These ten levels *can* be completed by a competent adult in about an hour.

After completing all of the tasks you then have some understanding of how the 'Blockly' interface works and now, when you click the 'Create' button at the bottom of the main screen, you finally get access to the full 'Blockly' programming interface with all of the block categories available.



Since this is written in 'Blockly', the blocks are a little different to those used in Scratch and mBlock 5, but the principles and concepts are very similar, and I have seen first-hand that a bright eight-year-old can very easily transfer the skills gained here to programming in mBlock 5 after using the 'Blockly' app.

'Blockly' dates back to 2011 and whilst not quite the same, visually resembles Scratch. It is a Google open-source library project for drag-and-drop block coding and typically runs in a web browser. It is primarily used for computer science education but also gives advanced users a way to write their own scripts for app creation. It has a very neat way of changing which blocks are shown in the toolbox palette on the left of the interface too. Blockly can be used to generate JavaScript, Python, PHP or Dart code and it can also be customised to generate code in any textual computer language.



Chapter 8 - All About Batteries

Powering mBot - what batteries should you use?

mBots box does state clearly that batteries are not included and mBot comes with a battery pack to which you need to add 4 AA batteries - it is shown in the instructions being attached to mBots chassis using 'Velcro' strips - we omitted these which sensibly allowed the battery pack to slide in-and-out from under the mCore board whenever we needed to change the batteries. The four 'Duracell Plus' batteries we inserted at build-time lasted about 4 weeks of daily use.

N.B. In addition to the above, the IR remote needs a CR2025 3V Lithium, button type battery (e.g. Panasonic CR2025) also not supplied in the box. These cost about 40p each.

...BUT on my mBot, low batteries were the cause of a rather worrying blinking LED and Tick-Tick sound problem (and possibly an 'iffy' Bluetooth connection problem too). All Arduino boards do blink a tiny LED light (marked by an L printed next to it on the board).
*Note: - **Everything does get sluggish (inc. your scripts) when the batteries are low.***

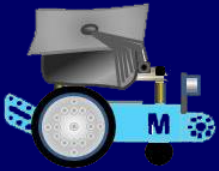
After about 6 weeks, I took the plunge and we switched to using a 3.7V rechargeable **Lithium Ion Polymer (LiPo)** battery which is very practical as the mBot board can charge the battery via its USB cable. These are thin, light & powerful and include integral protection circuitry.

I had thought of buying one from China via eBay since sourcing one of these in the UK seemed initially to be difficult; and what about getting the right connector? (the eBay pages were a bit vague on these) - it needed to be a 2mm pitch JST-PH connector (which is a 2mm spacing between the pins) - but then I found that the excellent and friendly 'Cool Components' in Hampshire had 2000mAh ones in stock for £15.50 (inc. speedy delivery - within 24 hours!).

Pundits in the USA advise a 2500 mAh LiPo, however 1000 or 1200 mAh batteries are quoted as being quite acceptable so I am expecting the 2000mAh one to work well. A small battery box (measuring 50mm x 33mm x 10mm) is supplied as part of the mBot components but this was far too small since the 2000mAh LiPo battery that I had bought was 54mm x 54mm x 6mm. This was not a problem since we just used one of the 'Velcro' strips omitted earlier to hold it in place under the mCore board - this is a good arrangement. *It's hard to understand Makeblock's reasoning for including a LiPo battery box but with NO information on purchasing a suitable LiPo battery to fit in it.*

When you plug the USB cable into mBot with a LiPo battery attached, one of two charging LEDs near the JST port light up. A steady red light indicates charging whilst a solid green indicates a fully charged battery. Apparently, if the green LED is flashing, it indicates a dead / bad LiPo battery. My new battery worked as prescribed and had some power in it, but it was fully charged after about 2 hours on USB.





Additional Facts about mBot using AA batteries & LiPo batteries.

1. **mBot** can operate from AA batteries (4 x 1,5V) or a LiPo battery (1 x 3.7V).
2. **mBot** needs +5V internally to operate. **mBot** power supplied via a TP3605 switching regulator (U1). This regulator will supply the correct voltage to **mBot** provided there is sufficient energy in either the AA cells or the LiPo cell.

mBot will cease to operate once the under-voltage threshold of the TP3056 is reached. The **mBot** motors are supplied directly from the battery, not via the regulator. This means that as the batteries deplete the motor speed will decrease.

3. **mBot** will not charge the AA cells connected to JP2 (barrel).
4. **mBot** will however charge a LiPo cell connected to PI (JST).
5. **mBot** has an onboard charger TP4056 (U2) that will charge the LiPo battery via the USB connector.
6. The onboard charger is programmed to charge the LiPo cell at a maximum current of 1 amp. This is interesting as most USB ports will *generally* only supply 500mA.

If you wish to charge the **mBot** LiPo cell in the shortest time, then ensure that you use a USB source that will supply more than 1 amp.

7. If the **mBot** on board charger has a > 1amp source connected, then charge time = capacity (mAh / 1000) hours. If connected to a standard USB port limited to 500mA then charge time = (mAh / 500) hours
8. The TP4056 chip has a battery over temperature cut-out but this has not been implemented by **MakeBlock**.
9. You may have AA cells and LiPo cell connected at the same time, but **mBot** will deplete the AA cells first and then switch over to the LiPo cell.
10. Measuring battery charge level using battery voltage will **not** provide a reliable indication of remaining capacity.
11. The **mBot** on-board charger (TP4056) is designed to trickle charge the LiPo battery at 130mA if the battery voltage is less than 2.9 volts.

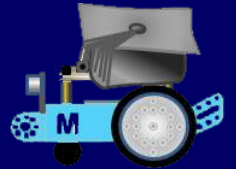
Once the battery voltage rises to above 2.9 volts the charger will switch to high current mode and charge at about 1 amp, provided the USB power source can supply 1 amp.

This trickle charge rate is designed to protect a over discharged battery. If the battery is over discharged, then the 130mA trickle charge can take many hours to get the battery up to 2.9 volts so that the TP4056 can safely apply the high charge current of 1 amp.

12. The **mBot** on-board power supply (TP3605) will supply +5 volts to the **mBot** until the battery voltage falls to about 3.0 volts at which point the power supply will no longer provide power to the mBot. This is done to protect the LiPo battery.

N.B. A 3.7volt LiPo battery with a terminal voltage of 3.0 volts is completely discharged. Taking any more energy out of the battery may damage it.

After a year of problem-free LiPo battery usage, I am so glad that I took the plunge & bought it.



Chapter 9 - mBlock 5 / Scratch - the Basics

mBlock 5 has its own robotics-specific programming blocks. These are additional to the standard Scratch 3 blocks, allowing each of the many 'Me' sensors and actuators available for Makeblock devices to be queried and controlled. You can also write sophisticated programmes for any Arduino based electronics device; but you don't however just have to programme robotics or electronics devices with the mBlock software. You can also use this software, just like its parent (Scratch), to write interactive stories, design games and create sophisticated visual effects.

With mBlock 5 software installed on your computer you can easily programme Scratch code sequences by just dragging, dropping, linking and fitting together colour-coded building blocks that represent control structures, assignments or actions.

In computer programming, a subroutine is a sequence of instructions grouped together (under a descriptive name) designed to perform a specific task. This named sequence can then be used, or 'called', in programmes whenever that task needs to be performed. In different programming languages, a subroutine may be called a 'Procedure', a 'Function' or a 'Macro'. A 'Script' is the Scratch word for such a subroutine which describes a collection or 'Stack' of blocks that all interlock with one another.

To create a script, you just drag blocks out of the 'Blocks' area and graphically assemble them as a series of connected blocks in the 'Scripts' area. A Scratch (or mBlock 5) script is defined as a set of blocks that begins with a 'Hat' block and therefore usually consist of at least two blocks.

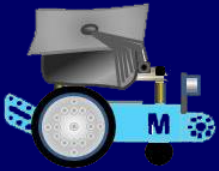
A script fragment is a block or set of blocks in the scripts area that is incomplete because it is missing a 'Hat' block. It is possible to run script fragment blocks by just clicking on them in the 'Blocks' area of the interface (as mentioned on page 28). This page also described a good habit to develop when clicking on programming blocks (to either drag them or activate them) - *always avoid clicking on areas on the block expecting further user input*. A script fragment will not run during normal execution of a project, because nothing triggers the code, but they do have their uses to set, or reset, things.

To start or 'Run' a script, just click on its 'Hat' block or press the 'Green Flag' button below the stage.

To remove a script, either right-click and choose 'Delete Block' or click and drag it over the 'Blocks' area where you will see it turn grey and a 'Trash Can' icon will appear - just let go of the blocks anywhere when this area turns grey.

The order of blocks in a script is very important as they determine how sprites interact with each other and the stage backdrop. It is considered good-practise to attach 'Comments' boxes to 'Scripts' to explain what certain blocks do and what the script's purpose is - I always try to add a comment to every script to describe its purpose and other comments for individual blocks if an explanation or reminder is necessary.

After you have created your scripts in the 'Scripts' area of the interface you can (but you don't have to) right-click the background and select 'Clean Up Blocks'. Doing so let's your scripts be organized automatically and neatly with their left edges aligned. This block alignment doesn't account for any comments boxes that you may add to your scripts and these may overlap after an auto clean up.



About Scratch Programming Block Shapes (as used in mBlock 5)

There are five shapes of programming blocks which can be connected to each other vertically to create 'Scripts'. Each data type has its own block shape and specially shaped slots, bumps and notches.

'Stack' blocks, 'Reporter' blocks and 'Boolean' blocks are all capable of dynamically resizing too and will stretch or shrink horizontally to accommodate other blocks or typed data inserted where individual values are needed. Every block shape is designed so that it can do one or more of the following:






Start a script.

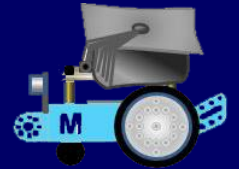
End a script.

Fit inside other blocks.

Contain other blocks.

There are six different block shapes and each type ('Hat', 'Stack', 'Reporter', 'Boolean', 'Wrap' or 'Cap') has its own shape and many have a shaped slot for other blocks to be inserted (or 'nested').

1. **'Hat'** blocks are the blocks that start every script. They are shaped with a rounded top and a bump at the bottom, so you can only place blocks below them. Each Hat block type is activated by a specified method, enabling different scripts to be started at different times. The general shape of a Hat block is shown on the right:

2. A **'Stack'** Block is a rectangular block that can fit above or below other blocks with a notch at the top and a bump on the bottom for connecting to other blocks. 'Stack' blocks make up most of the blocks available in every category except in the 'Operators' block group. They perform specific commands and when two or more stack blocks are connected to form a 'Script' then their individual commands will execute in sequence order from top to bottom. The shape of a typical stack block is shown on the right:

3. **'Reporter'** blocks are elongated blocks with rounded ends and are designed to fit into, (nested inside) matching shaped slots in other blocks wherever a value (either a number or a text string) is needed. They have no bumps or notches and can't therefore can't be used alone. To quickly view the value of a reporter block, simply click it and a pop-up bubble will display its value. A typical Reporter Block is shown on the right:

4. **'Boolean'** blocks are elongated hexagonal blocks, shaped like the Boolean elements in flowcharts and they fit into corresponding hexagonal slots in other blocks; they too have no bumps or notches and therefore can't be used alone. A Boolean block contains a condition, which can be either 'true' or 'false' or the numbers '1' and '0' depending on their usage in a script. A common use for a 'Boolean' block is within an 'If ... Then' 'C' block. A typical Boolean Block is shown on the right:

5. **'C'** blocks are also sometimes known as 'Wrap' blocks. These blocks *loop* any blocks contained within the 'C' checking if a condition is true before exiting. There are five types of 'C' blocks, found in the 'Control' category. 'C' blocks can be either bumped at the bottom for further connecting, or flat (capped).




6. There are two specific **'Cap'** blocks - *'Stop all'* and the *'Forever'* loop *'C'* block to be found in mBlock 5 in the *'Control'* category. Cap blocks are used to terminate scripts. They are shaped with a notch at the top and they have a flat bottom so you cannot place any blocks below them. The general form of a Cap Block is shown on the right:



In addition to the standard Scratch/mBlock 5 blocks listed above, there is also the facility to create very useful **'My Blocks'**. These are user-defined *'custom'* blocks, a way of compressing a script containing a sequence of blocks into a single stack block that you name yourself. A *'My Blocks'* hat block is shown here on the right:



You can perhaps perceive a *'My Blocks'* block as an alternative to a hat block since they are always needed to start any script sequence of programming blocks.

N.B. They are only visible in (and work in) the sprite (or the device) in which they are created!

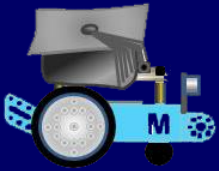
It's considered best practice not to have repeated chunks of code in scripts, so using a self-defined *'My Blocks'* block only need a single custom stack block to call any code created under it. These blocks can be used in scripts as many times as you like and save much duplication of block sequences since you only need to use the name of the custom block every time you want to call the common code; this will reduce the total number of blocks (and therefore the parsing time) in your scripts.

mBlock 5 needs to parse the entire set of blocks in your *'Scripts'* area; so breaking up one big script into smaller scripts is generally a good idea as it reduces the parsing time making your code run a little faster and smoother. As well as saving time and saving space, using well-named custom stack blocks can also provide much clarity in a complex programming sequence making your programmes easier to read and to maintain - so *do* develop the habit of making your own custom blocks.

Basic Programming Principles

When you know what you are doing (*and have a real requirement to do so*) you can upload the code from mBlock 5 via its integrated compiler into the flash memory of your robotic device (mBot) so that it can run repeatedly and totally independent of your PC in *'Offline mode'*.

Uploaded code is actually more efficient and runs more smoothly when all your instructions are running totally inside mBot's brain since the mBlock 5 application is really a code generator. If you do upload your scripts into mBot it compiles your programme into a hexadecimal source file (that conveys binary information in ASCII text form (this is a method commonly used for programming microcontrollers, EPROMs, and other logic devices). Conversely, with the more usual *'Online mode'* control of mBot you might experience marginal lag times because of the emulation happening to bridge your code with mBots hardware. *However, for most of the time, there is no need go 'Offline' at all* - it is rather boring and somewhat tedious to upload your code into mBot which is then stuck with just that set of instructions until the equally tedious *'Upgrade Firmware'* option is invoked. You may also find it worth it to use *'Reset Default Program'* to clear out mBots memory too. So do get into the habit of always working in *'Online mode'*, allowing mBot to provide real-time development feedback on the *'Stage'* using *'Monitors'*, *'Variables'* and other graphical output from your projects (see Chapter 14 for more on graphics).

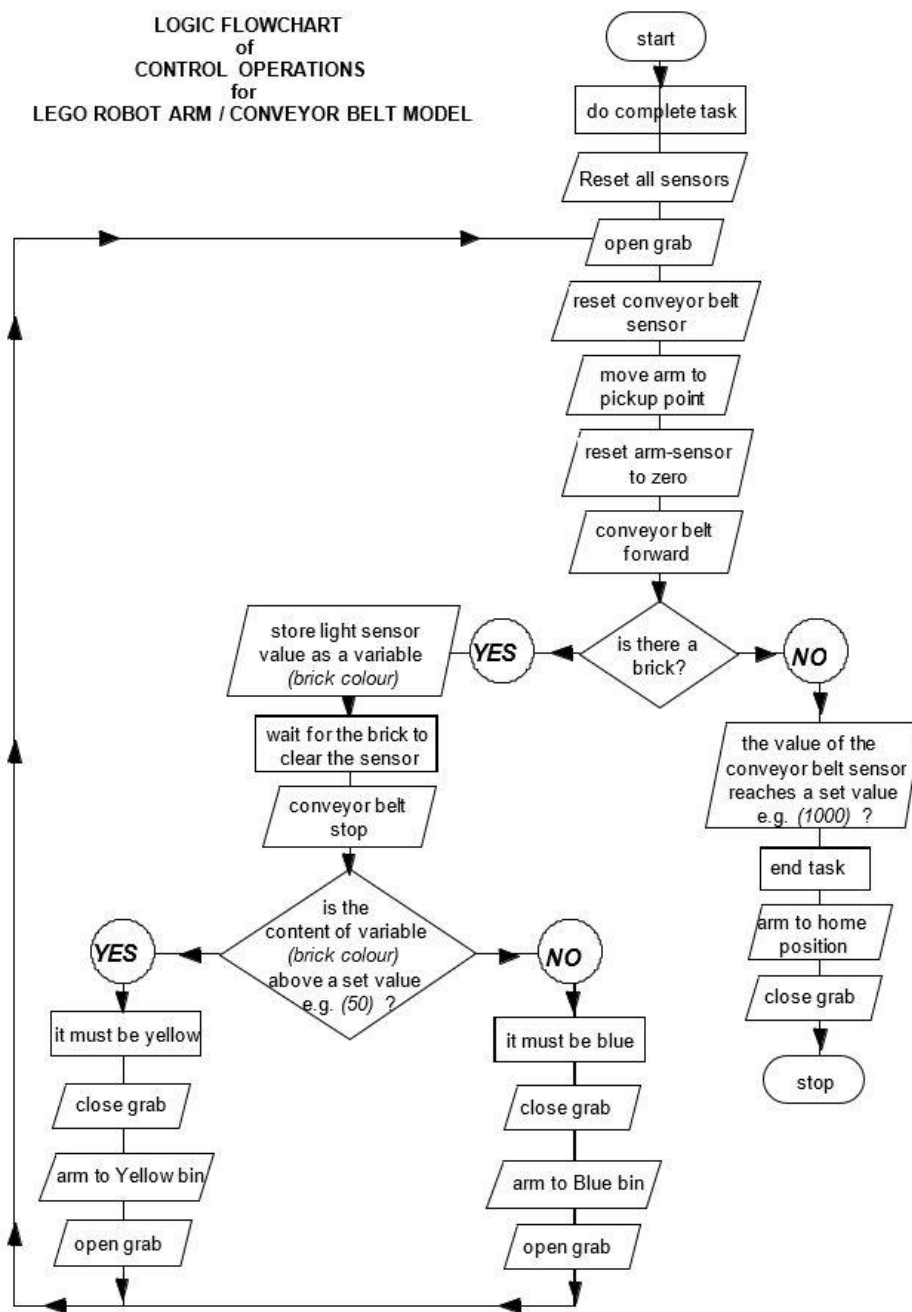


mBlock 5 lends itself to simple procedural coding since there are no complex functions, no multi-threading, no fancy data structures and only the simplest manipulation of variables. Ideally, you just work out a decision-making algorithm detailing the *Input / Process / Output* of what you want to do; and then write the script - simple!

N.B. An algorithm is a sequence of step-by-step instructions or a set of rules that are followed in the right order to complete a task correctly.

An algorithm is a procedure that tells your computer precisely what to do (and in what order) to solve a problem or reach a goal. The task to be solved by an algorithm can be anything, so long as you can give clear instructions for it and it can be written as a list of steps just using text or mostly, pictorially (a flowchart) with shapes and arrows showing input, processing and output.

LOGIC FLOWCHART
of
CONTROL OPERATIONS
for
LEGO ROBOT ARM / CONVEYOR BELT MODEL

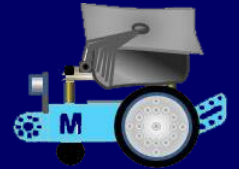


Shown on the left as an example of this is a complex decision-making algorithm that I created about twenty years ago showing the sequence required to operate a Lego model using the 'Logo' programming language.

It's a good habit to always try to sketch out a quick algorithm like this to work out the steps you need to achieve a programming goal and you will soon work out a way to build scripts that do exactly what you want.

As in all things, keep your programming steps simple and name *everything* that you add into a programme yourself (Variables, Lists or Blocks) using meaningfully descriptive names. Such names should be immediately understandable as part of a programme sequence.

At its simplest level, an mBlock 5 script can be a linear sequence of simple stack blocks with a 'Hat' block at the top to enable the sequence to be activated (see the diagram on the next page).



Each block in a script carries-out its designated command in turn in the order in which the blocks are positioned, from top-to-bottom, and then the sequence stops.

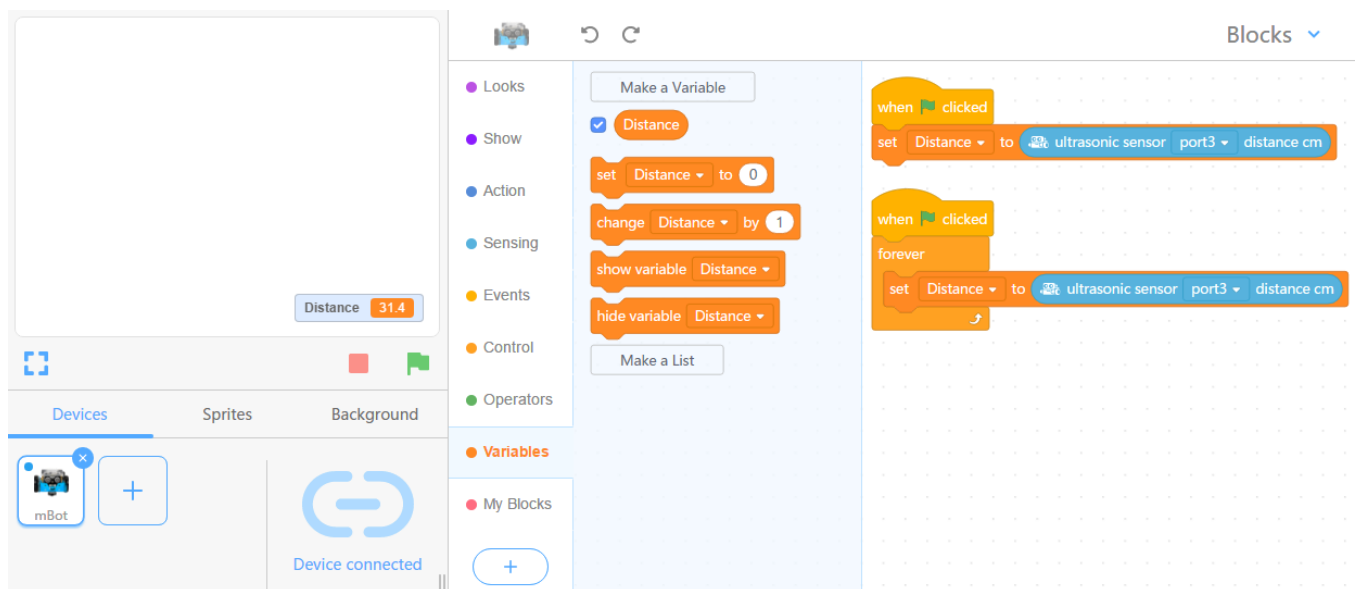
It is fairly easy to understand and modify what is happening in a linear block sequence; but it is a 'one-hit' action, and since there is no decision making, it needs to be activated by you if you want the script to run again.

The next level of programming, moving on from the simple linear sequence shown on the right is to use looping (decision making) sequences. You will use looping sequences for much of your programming.

A good example of needing a looping script is for polling a sensor such as mBot's ultrasonic sensor 'ears' to display its distance from objects; and then storing the feedback in a user-defined 'Variable' called 'Distance'. If you make a linear block script to show feedback from the sensor by putting it into a named variable then it's monitor window can be displayed on the 'Stage'- but it will only show a number which was the distance measured by the sensor when it was polled by running the script; and the number will not change in real-time.



However, if the same Reporter block (nested inside a Stack block) is put inside a 'Forever' loop, then the number on the displayed variable will constantly change as soon as the script is run; this is what is known as 'real-time' feedback! - see the screenshot of both of these scripts below:



In mBlock 5, looping blocks are 'Control' blocks which hold other blocks inside them. The most common of these is 'Forever' which is a loop block (shown in the screenshot above) that repeats forever in an infinite loop; and there are surprisingly, a lot of cases when an infinite loop is needed. Using a 'Repeat' block instead will loop a given amount of times before allowing the script to continue.

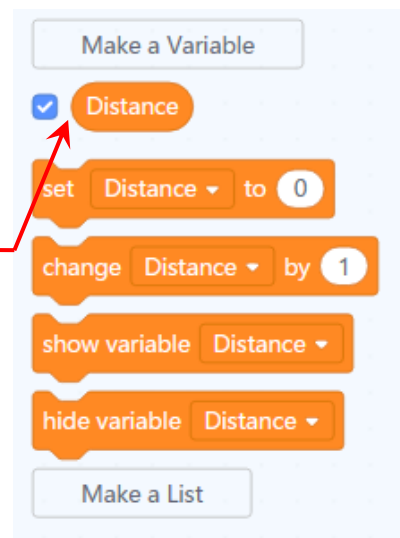


If you put one 'Repeat' block inside another (which *is* allowed) this is also called 'nesting'. The total repetitions will be the product of the two nested 'Repeat' block inputs.

The 'Repeat Until' block is very similar and will loop until the specified Boolean statement is true, in which case the code beneath the block (if any) will execute. Note that this loop is in similar nature to a 'While' loop in some other programming languages.

Particularly useful and probably the most powerful of the looping blocks are the ones that make decisions after checking specified conditions such as 'If...Then' and 'If...Then...Else'; both of these conditions loop until the specified condition is true and then exit the loop.

The example described on the previous page, used a user-defined 'Variable' which I named 'Distance'. In programming, a variable is something created to hold a value, much like x and y are used as variables in algebra. In mBlock 5, variables are represented by 'Reporter' blocks shaped like long rectangles with semi-circular ends and when created must be, uniquely labelled by you - note the uses of the 'Distance' variable in the diagram on the right.



If you define a variable, then several more stack blocks appear under the heading 'Variables' in the block categories section of the 'Blocks' area. They are orange in colour and as you create more variables then their names are automatically added to the drop-down menu lists in each of the blocks. The current choice is shown as 'Distance' in all the blocks shown above right.

Variables, generally speaking, can be local or global. In mBlock, a local variable can be used by just one sprite; however, a global variable (the default setting) can be used by *all* sprites.

One of the great strengths of mBlock is the ability (and a requirement in many cases) to 'dock' or 'nest' programming blocks inside other blocks. The example on the previous page using the variable 'Distance' allowed a choice from four additional orange stack blocks. The first block in this list; shown slightly more clearly above is 'set (variable name) 'Distance' to '0'.

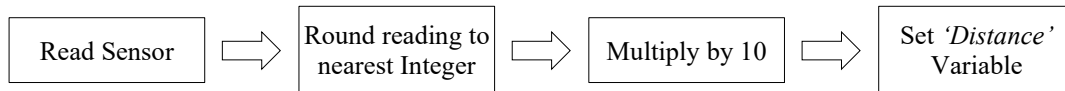
I used this stack block to create both of the single block scripts in the example shown on the previous page. Note that the '0' part of this can be changed to anything that you can type **or that can be nested inside the window that contains '0'**.

In this case, the 'Reporter' block for the Ultrasonic Sensor was dragged over the little window until it 'nested' inside it (be aware that the rim of the window *illuminates* with a white rim when another block is near enough to be nested inside it) - note too that the 'Set (Variable Name) to...' block also dynamically expands automatically to allow any nested inclusion to fit inside it. Blocks grow clearly wider when something is nested inside them, but blocks grow a little taller (or fatter) too with each subsequent nesting.

With a little practice, nesting mathematical operators and reporter blocks to calculate meaningful output becomes very straightforward. As an engineer, I like to have distance readings in millimetres (not the default measurement, centimetres!).



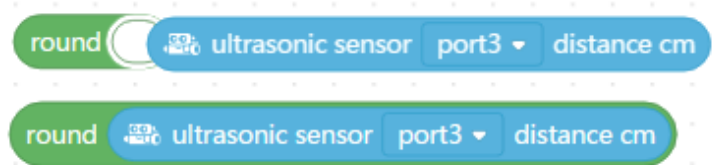
To do this (as shown below) it needs a slightly more complex 'nested' variant of the same single stack block. First, the sequence is described as a simple algorithm:



Next, here are the blocks required to achieve this:



Build the nested block sequence in the following order – note that the window in the 'round' operators block is illuminated to show that the 'nest' can be made (and note too how the operators block stretches to accept the 'reporter' block):



Add the newly nested 'round' block into the left-hand window of the 'multiplication' operators block and type '10' into the right-hand window as shown below:



Finally, add this nest of the three component parts to the round-ended right-hand window of the Set 'Distance' Variable stack block to complete the block sequence required to allow 'Distance' to report distances in whole millimetres:

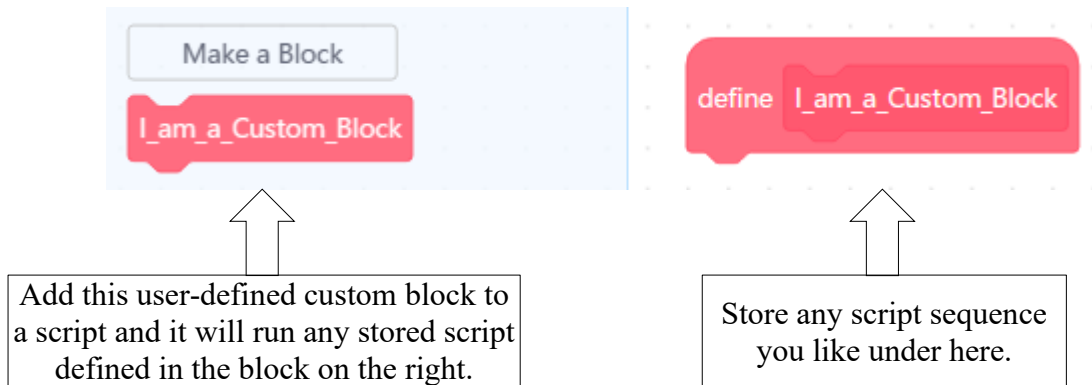
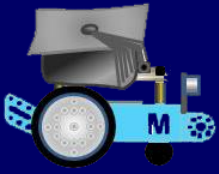


As mentioned earlier, it's considered best practice not to have repeated chunks of code in your scripts, so you need to develop the technique of making your own 'user-defined' custom blocks which you create by clicking the heading 'My Blocks' at the top of the block categories section of the 'Blocks' list area.

These blocks are *SO* useful!

Named custom blocks that you create here are red in colour and greatly help with the recurring use of script sequences in a complicated programme since you only need to use the name of the custom block every time you want to call that piece of code.

After defining a block and creating a script under the (unique to 'My Blocks') red coloured hat block shown on the right of the diagram at the top of the next page, a new stack block with the name you gave to the hat block is available and visible in the 'My Blocks' window enabling you it to add to any other script sequences as many times as you like - one block such as this can call the whole script stored under its defined hat block; saving space and providing much clarity in a complex programming sequence.



A Little More Detail About Variables

It is helpful perhaps to consider variables as being containers that hold information; with their sole purpose being to label and store data in memory which can then be used throughout a programme.

But although variables holding 'Device' data can be seen on stage output monitors, any scripts created on the 'Sprites' tab can't actively use any of this data in their programmes without using the 'transfer-data-on-demand-by-message' broadcasting trick described in Chapter 13 (on pages 77 to 79).

In programming, a variable is a value that can change depending on conditions or on information passed to the programme. Storing bits of data is crucial to the majority of programming scripts. In mBlock 5 data can be held in either variables or lists and either of these can be used whenever alphanumeric or boolean values need to be stored and then referenced and manipulated in a script.

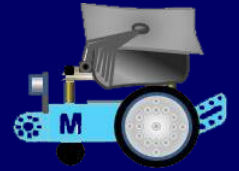
The naming of variables is regarded as one of the most difficult tasks in computer programming and you need to develop a method for labelling data with a descriptive name so that your programmes can always be understood clearly.

When you assign names to variables, try to make sure that the names are accurately descriptive and sensibly logical. They should be understandable to either someone else or indeed yourself when you return to a programme that you may have scripted months *or even years* earlier.

Creating variables in mBlock 5 is not difficult - in fact it is a fairly simple but marginally tedious task (if you do need to create many of them), so having a set of useful and logically named variables ready-made in a setup programme is a good idea.

In addition to creating useful graphics libraries and having them pre-loaded into my '*mBot Setup Page*' file (previously mentioned on page 21) I have also experimented by added to it my most frequently used variables, especially those which are needed to enable any of my installed library graphics to be programmed on mBlock's stage.

A list of these variables is shown on the next page. Any project based on a set-up file such as this can contain all of these (even if they are not being used) and it is worth noting that both the graphics and the variables pre-loaded into a file such as this are *much quicker to delete if they are not needed in a project than they taken to create them!* HOWEVER, I have reverted to no longer having any pre-loaded variables in my setup file.



Why have I inserted an underscore (_) in these Variable Names?

When creating any project that is likely to be uploaded to Arduino, it needs meaningful names throughout and especially for variable names, broadcasts and self-defined block functions.

These should ideally be just one descriptive word or one continuous set of characters e.g. (using an underscore to simulate a space).

However, if your mBlock projects are not ever likely to be uploaded to Arduino, then spaces are OK and continuous characters (although a good habit to adopt) are not important.

- Counter
- Digit_Delay
- Digit_Hundreds
- Digit_Size
- Digit_Spacer
- Digit_Tens
- Digit_Units
- Digit_Xpos
- Digit_Ypos
- Digits_Shown
- Lamp_Pos
- Monitor_Xpos
- Monitor_Ypos
- Sensor_Pos
- Sensor_Value
- Stage_Xpos
- Stage_Ypos
- Switch_Pos

About Lists

mBlock's Lists (called arrays in other programming languages) are a way of storing multiple pieces of information at once.

A 'List' can store and hold data that can be retrieved, added to or acted upon by other scripts.

A 'List' can also be defined as a variable containing multiple other variables. Data stored in lists in mBlock 5 can be usefully exported as a text file and text files can just as easily be imported to populate an mBlock 5 list.

Lists are in effect, a simple vertical lookup table which consist of two vertical columns the first containing ascending numbers which are paired with a second column containing alphanumeric data; each data item capable of being retrieved by calling its paired number. When you add a new data line, then a new pairing number is auto-created.

```

when space key pressed
  set Random_Number to pick random 1 to 3
  set Text_1 to Oh! Emma
  set Text_2 to item Random_Number of Useful_List
  
```

This example uses a list to store the data required for a very simple 'Rock, Paper, Scissors' game.

Useful_List

1	Rock
2	Paper
3	Scissors

+ length 3 =

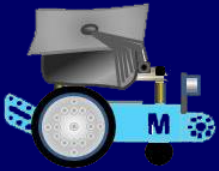
Text_1 Oh! Emma

Text_2 Scissors

You can see from the short script above how a randomly generated number can be used to look-up items stored in the list shown on the right to enable mBlock to return a text response every time Emma pressed the 'Space' bar on her keyboard with one hand whilst making her own hand gesture with the other hand at the same time.



This was fun to do, and we made, as you will see later on, many more sophisticated variants of this game.



Chapter 10 - About mBlock 5 - in more Detail

It is important to grasp at this stage, the concept of mBlock 5 having in reality two different Scratch programming sections to use (with mostly different blocks in each of them). Once you understand this then it is easy to see where you are going to create '**Device**' scripts that control mBot (which you will inevitably start-off doing initially anyway) and then eventually tackling rather more complex '**Sprite**' programming scripts that can be used to animate useful graphics.

All about mBlock 5's Default Set of '**Device**' Programming Blocks for mBot

You do need to understand that **you can only programme your chosen robotic device (mBot) when the '**Devices**' tab is selected** and that initially, you only have access to the default set of robotics blocks available in the blocks area. Slightly more complex programming however requires more blocks than these to create control scripts and these are added as '**Extensions**'. Robotics extension packs are only available to '**Devices**'.

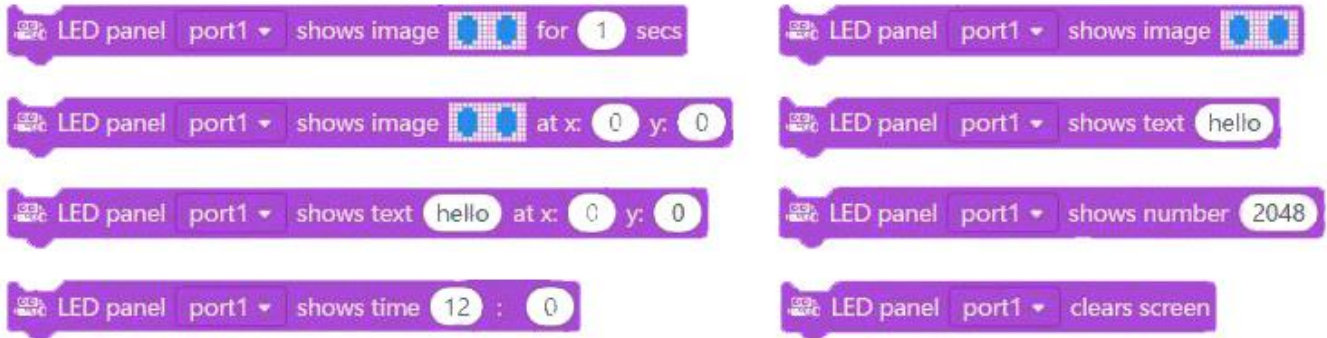
Unlike all previous versions of mBlock, '**Sprites**' no longer support robotics block programming and the extensions packs of robotics blocks cannot be added to '**Sprites**' - but more on extensions later. It is important to realise too that **when you switch to the '**Sprites**' tab, the categories in the '**Blocks**' area change completely to show a totally different set of (non-robotics) programming blocks**. As mentioned earlier (on page 18) the '**Blocks**' area has a set of coloured category buttons where you can choose from sets of programming blocks that you need to build control scripts. There are nine categories of these default blocks specifically for the mBot device and when you click on any of these buttons then the available programming blocks in that category will be visible (short blocks anyway) in the sub-section immediately to the right. Clicking anywhere on this sub-section panel allows longer blocks to flow out over the '**Scripts**' area to show their full content.

Although the block categories remain the same, or very similar, when you change devices, they do vary for each robotic device depending upon the capabilities of each. '**Codey Rocket**' has, for instance, eleven categories of robotics blocks to access whilst '**HaloCode**' & '**Ultimate2**' robot have nine and '**mBot Ranger**' only eight.

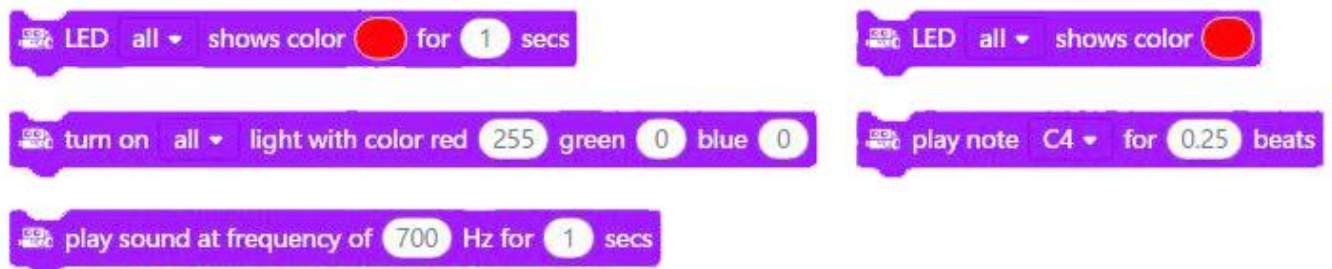


The first four of mBot's block categories are '**Show**', '**LightSound**', '**Action**', and '**Sensing**' and all of the blocks in each of these have a device image at the left hand end of the block (a lovely new mBlock 5 design feature). In the image on the left you can clearly see that it's an mBot-badged block and coloured blue indicating that it's from the '**Action**' block category.

The first category of these mBot-badged blocks is '**Show**' and although it is the first category group, it has little to do with initial mBot programming since these eight stack blocks are all for creating scripts to operate Makeblock's add-on component, the '**Me LED Matrix Panel**' (known in earlier versions as '**Face**'). This is an 8 x 16 matrix of little LED lights (more about this in Appendix 1 on page 184) - it is a very useful output component, but not part of the basic mBot robot! I suspect that the '**Show**' category comes first in the blocks categories list to maintain compatibility with '**Codey**' categories since '**Codey**' has an inbuilt LED display; but if you want LED matrix output from mBot, then you have to buy it separately (but it is worth it!). The '**Show**' blocks are shown at the top of the next page.

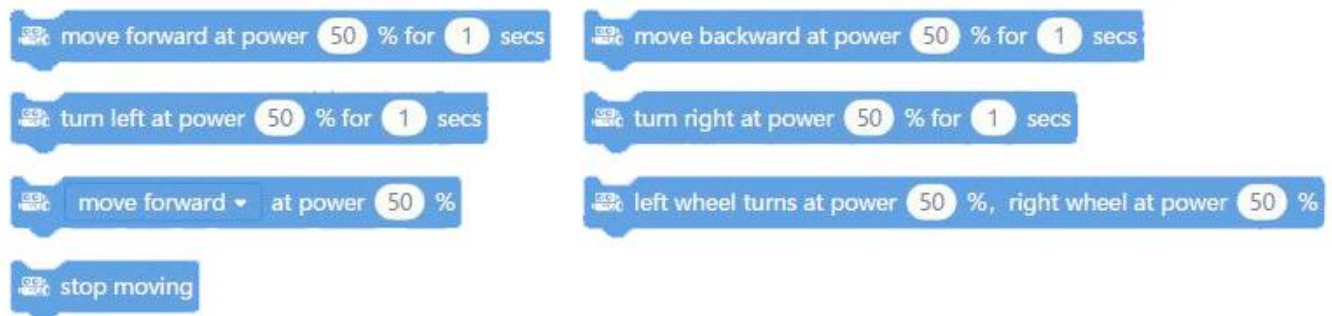


The second button in mBot's 'Blocks' categories list accesses the 'LightSound' blocks. The five 'stack' blocks in this category are itemised below:

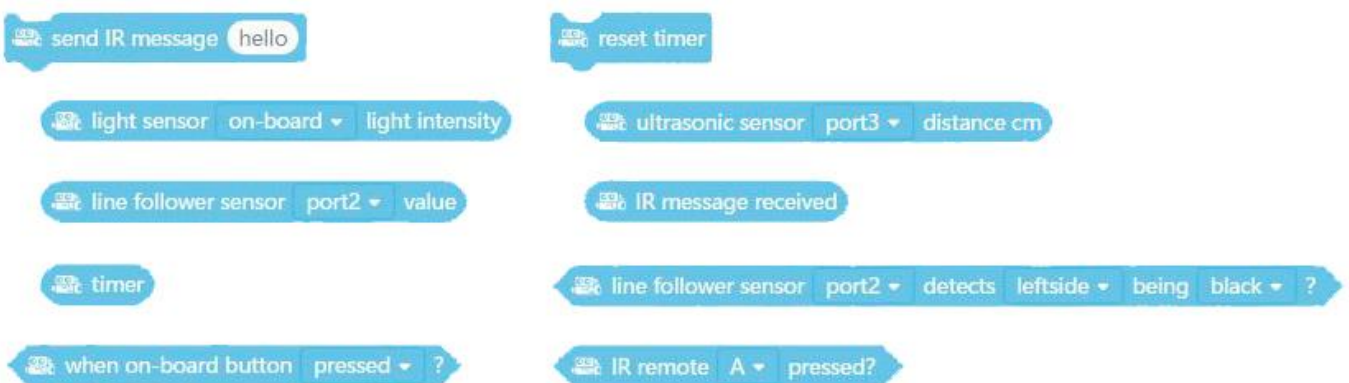


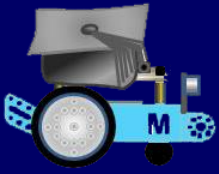
With these blocks you can write scripts to control very simple output from mBot using the coloured LEDs and the sound generator buzzer on the mBot's *mCore* board.

The third button in mBot's categories list accesses the 'Action' blocks. The seven 'stack' blocks in this category (shown below) allow you to write scripts to control the two electric motors that power mBot.



The fourth button in mBot's categories list accesses the 'Sensing' blocks. These blocks are different in appearance to those in the previous categories shown above. There are ten blocks in this category which are shown below:





mBot and Me a Beginner's Guide

Out of the ten 'Sensing' blocks, there are only two 'stack' blocks, the remainder being 'reporter' or 'boolean' blocks. Four of the five 'reporter' blocks have a check-box, which if ticked, will show the reported result (feedback from that reporter) on mBlock's stage in the same way that 'Variables' can.

The final three blocks in this category are 'boolean' blocks shaped to fit into matching hexagonal holes in 'control' or 'operator' blocks to enable decision making within that block. Below the first four mBot block categories described here and on the previous page are three more categories which do not carry the 'mBot specific' badge described earlier.

These categories are the 'Events', 'Control' and 'Operators' blocks which can be considered as 'general-purpose' programming blocks (all very similar to their Scratch 3 counterparts). Blocks in the 'Events' category comprise the five 'hat' blocks and the two 'stack' blocks which are shown below.

Crucially, there is no longer (as there was in mBlock 3) a 'when key released' hat block in this category; there is now only a 'when key () pressed' hat block.



There is not a 'when key released' hat block in Scratch 3.0 either.

Perhaps this is where this omission originates?

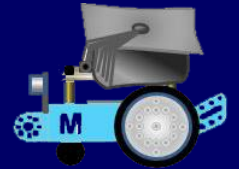
The 'when key pressed' hat block is pointless on its own since it performs in exactly the same way as the more usual 'when green flag clicked' hat block - namely starting a script action with no way of stopping it without clicking the 'stop' button below the stage or clicking the same 'hat' block again.

The 'when () key pressed' hat block still provides a useful way of running a script by pressing a designated key on your keyboard; but since there is no 'when key released' hat block **you can no longer hold down a key (i.e. 'the space bar') to run a script and then exit the script as soon as you release the key** (as you could in mBlock 3).

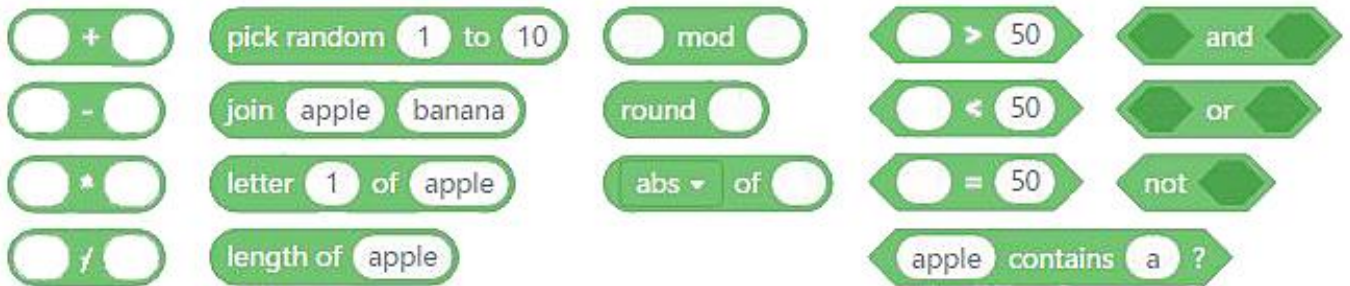
This key pressed / key released combination was, in the past, important for simple robotics control - a serious omission error here by the MIT Scratch team surely?

The eight blocks in the 'Control' category are identical to their Scratch counterparts. The three 'stack' blocks and five 'C' blocks in this category are shown below:





There are eleven 'reporter' blocks and seven 'boolean' blocks in the 'Operators' category, and these are very similar to their Scratch counterparts too. These blocks are shown below:



The final two mBot programming block categories are 'Variables' and 'My Blocks'. Neither of these categories contain any blocks on start-up because you have to create your own block as you need them.

About the Robotics 'Device' Programming Extensions Blocks for mBot

Pressing the (+) icon at the bottom of the 'Blocks' panel (beneath the coloured block category buttons) opens the 'Extensions Centre'. **This window shows totally different contents depending upon whether you have the 'Devices' tab active or on the 'Sprites' tab active.**

In addition to the default robotics blocks for programming devices listed in the last two pages, there are twelve extensions the 'Extensions Centre' when it is opened from the 'Devices' tab. You can however only see eight and it's not very clear that the window needs to be scrolled to see the remaining four.

Four of them: 'Light Sound', 'Servo', 'Sensing Gizmos' and 'Gadgets' contain between them just 11 different robotics blocks (*with some rather confusingly being repeated in these extensions three times!*).

However, these 11 blocks plus another 26 blocks (making 37 different blocks in total) are all contained in just **one** extension, the '**Maker's Platform**' extension. This is almost certainly the only extension that you ever need to load for the majority of your projects. So why ever bother loading the other four listed above? If you do ever have the need to load these then it is worth noting that the 'Light Sound' extension has robotics programming blocks for LED control and light and sound sensor feedback.

The 'Servo Pack' also has the same blocks for LED control in addition to those for controlling servo motors. The 'Sensing Gizmos' extension has blocks for basic electronics work inc. fan control and sound and temperature sensor feedback whilst the 'Gadgets Pack' extension also has the blocks for LED control and for controlling servo motors as well as limit-switch control and digital display output.

I'm not too sure yet where I stand on the usefulness of 'Extensions' - either for the 'Devices' tab or for the 'Sprites' tab especially since the remaining seven extensions for the 'Devices' tab are (for a variety of reasons) of very limited use.

I'm not sure if the 'Data Chart' extension is of much value at all - it links to 'Google sheets' but it's only available if you are using the mBlock 5 web browser version. The list of blocks in this extension appear to be very limited in what they can load or edit. The 'Upload Mode Broadcast' extension enables a device to interact with a sprite when it is used offline - occasionally useful, but generally a very rare occurrence!



mBot and Me *a Beginner's Guide*

The *'Bluetooth Controller'* extension enables the connection to a hand-held games type controller via Bluetooth. This might be of value to enable direct control of mBot if you already have one, but you can do that already with the little IR controller supplied with mBot. There are just two blocks in this extension, and they are greyed-out (dimmed) until a device is connected .

The next three extensions require you to spend money!

Specific Makeblock Me module boards (*rather than the more commonly available add-on packs*) are not an easy thing to source in the UK and at an average cost of £15 to £20, plus considerable shipping costs are rather expensive to source.

The *'Audio Player'* extension is for advanced recording, manipulation & playback of external sound files. BUT this needs a specific Me audio playback module if you want to play or record music or to add speech recognition to a project.

The *'RGB Line Follower'* extension like the *'Audio Player'* above needs the purchase of another specific Me module. It can be used for advanced robotic line-following, allowing mBot to be programmed to follow different coloured tracks. The *'Color Sensor'* extension (like both the *'Audio Player'* and the *'RGB Line Follower'* also needs the purchase of a specific Me module which is capable of determining between six different colours.

I'm not sure what the final extension is or does! - Its not currently activated so you can't open it and it *might* be linked to an **Internet Ordering Platform?**

I need and therefore regularly use the aforementioned *'Maker's Platform'* extension and I always load it into the edit page of all of my projects since it contains many of the useful blocks which were always present in the good old mBlock 3 *'Robots'* blocks category.

Once loaded into (and used in) a project, any extension such as this one is remembered and will be visible in the *'Devices'* blocks categories (or *'Sprites'* blocks categories, if appropriate) when that project is reopened.

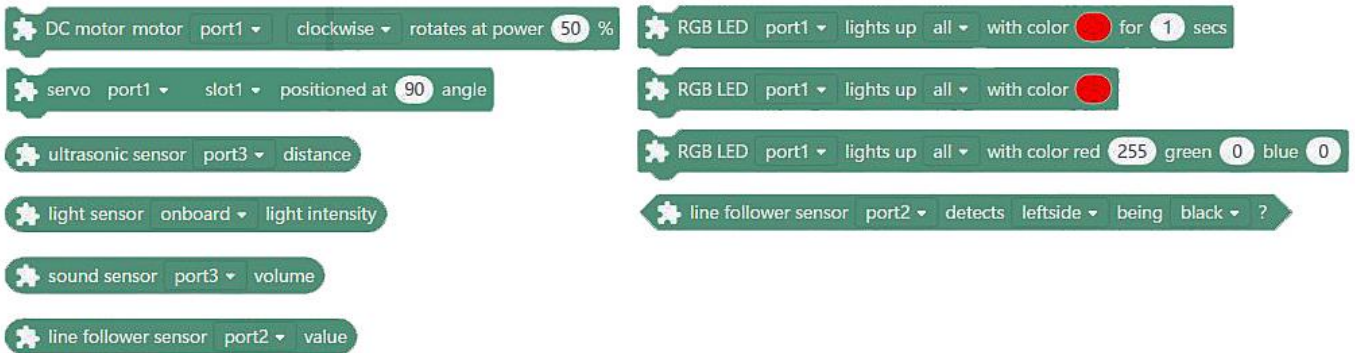
I use the *'Maker's Platform'* extension specifically for its DC motor control blocks which are much more precise for moving mBot compared to the rather simplistic, *'Move'* blocks in the *'Action'* category of mBlock 5's default sets of blocks.



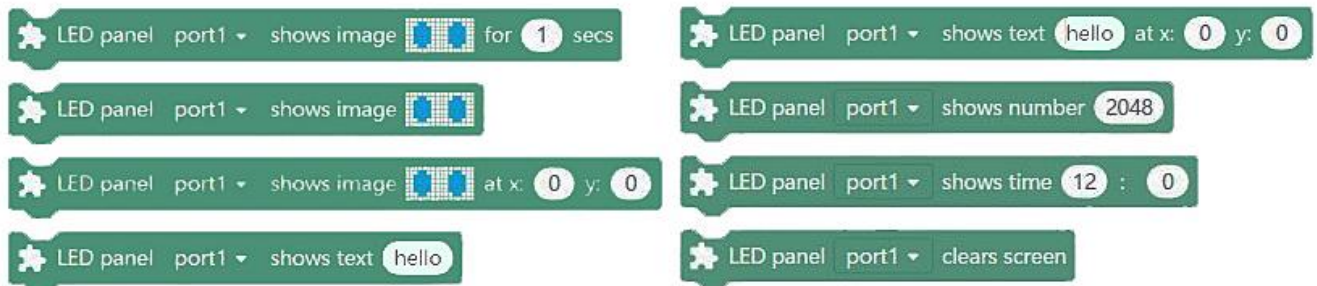
These DC motor control blocks enable you to run **one motor alone** or **independent of the other** and each can be set to run clockwise or anticlockwise - confusingly the descriptors on these control blocks refer to DC motors connected to either *port1* or *port2* - this does not however mean the main connection ports on the *mCore* board (*it really means the M1 & M2 motor connections*)!



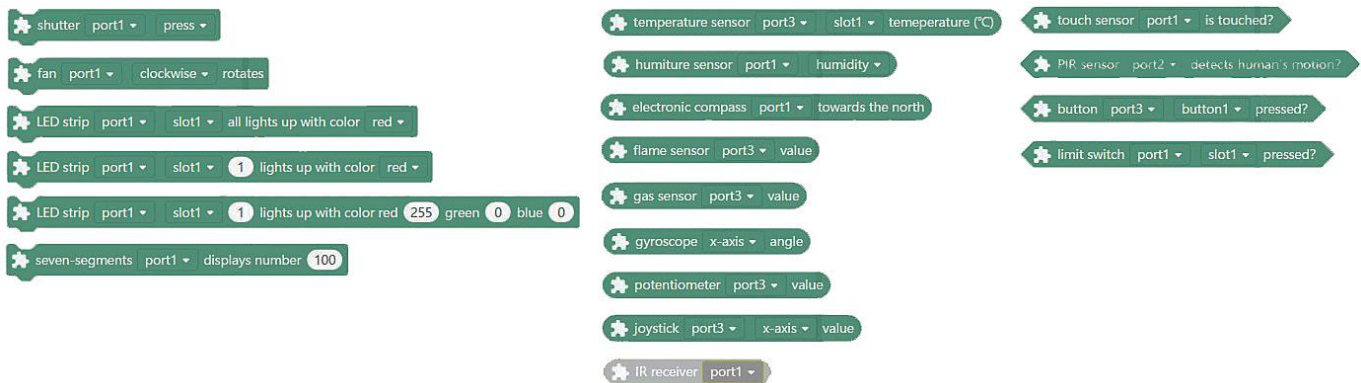
Out of the rather long (too long in fact) list of **34 blocks** in the *'Maker's Platform'* extension the following five stack blocks, four reporter blocks and a single boolean block are probably of the most use:



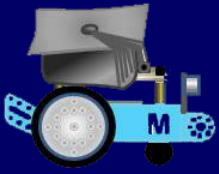
These are followed in usefulness by the following (*although these blocks all do exist in the 'Show' category*) - and are only of use if you buy the LED panel add-on component:



The remaining nineteen blocks in this extension are really only of use if you have access to specific control / feedback items which are (as mentioned on the previous page) expensive add-ons not commonly available:



N.B. Rather confusingly, the *'Light Sound'* 'Devices' tab extension mentioned on page 47 has the same name as the second of the blocks categories on the 'Devices' tab which is also called *'LightSound'*.



About the Stage 'Sprites' Programming Extensions Blocks for mBot

Fourteen individual sets of robotics blocks (totally different to those discussed on previous pages) are available for mBot in the 'Extension center' when it is opened from the 'Sprites' tab and once again I'm not sure about the real value of very many of these. However, that said, **this software is totally free to use and Makeblock have made a very good and nobly intentioned attempt at providing the tools to enable kids to get to grips with the basic concepts of A.I and Machine Learning.**

These individual 'Extension' groups are as follows:

The 'Cognitive Services' extension has Artificial Intelligence (A.I.) blocks with an integral on-screen 'Recognition Window' showing what your web-cam can see. The blocks work very well in recognising spoken and written words (both printed & hand-written). There are also blocks which purport to recognise the age and the emotional expressions of an individual viewed by the web-cam - I'm not too sure about the value of these last two but experimenting with simple A.I. feedback is quite good for kids. Technically (and rather grandly), 'Cognitive' refers to attention span, memory and reasoning; along with other actions of the brain that are considered to be *complex mental processes!*

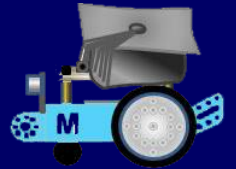
The 'Teachable Machine' extension has initially just one block which accesses a 'Training Model' window which enables your computer to *supposedly* learn things by establishing *artificial neural networks that resemble human brain processes* and which can train your sprites to 'learn' things - this is called 'Machine Learning'. This also has an integral on-screen 'Recognition Window' showing what your web-cam can see. This extension is rather complex to use, and the model relies on you creating 'categories' (rather like variables) which, when the model is in use, will become available as drop-down choices in two of the three new blocks added to the extension. These 'categories' are 'trained' on what the web-cam can see, which seems to be rather similar to the A.I. emotions feedback in the 'Cognitive Services' extension; but here, it doesn't appear to work half so well.

The 'Data Chart' extension provides a very simple way of building a data table, a bar or line chart on screen which can be downloaded as a .png file; but why? - This seems to serve no really useful purpose unless you want to use it to capture the data stored in variable reporter blocks in a graphical way.

The 'User Cloud Message' extension can be used to sync data from your mBlock account across different devices and projects - this is probably useful when you need to upload a programme to a device like 'HaloCode'. The 'Pen' extension is useful since it allows sprites to draw lines on the stage display screen. The 'Music' extension has its uses too if you want mBot to produce sounds emulating a variety of instruments (and this now works *so* much better than the 'Sound' blocks group did in mBlock 3).

The 'Climate Data' extension does have some value too and has robotics programming blocks for accessing and manipulating real-time climate data from most towns in countries across the world. The output from these blocks is rather sparse and ideally needs to be concatenated with descriptive text either side of the output data; which leads to some very long nested block sequences (but if these are stored as a named variable then the data string can then be called by just the variable name).

The 'Upload Mode Broadcast' extension is exactly the same extension as is found in the 'Device Extensions Centre'. This also enables a device to interact with a sprite when it is used offline - and as mentioned before, useful - but a very rare occurrence!



The *'Google Sheets'* extension can connect mBlock 5 to a Google spreadsheet of data, but only if the sheet has been *'Shared'* and anyone with the link allowed to edit the file. You need to have copied the link (URL) from the sheet you want to access to the clipboard so that you can paste it into mBlock 5's *'connect to shared sheet'* block.

There are only two additional blocks in this extension - one to input data into a named cell, and one to read data from a named cell. I suppose that you might want to use this as a way of capturing or inputting data stored in variable reporter blocks; but this seems to be a tedious and possibly pointless undertaking, so I've not tried that yet!

N.B. In a short test of this extension, it took about 8 seconds to poll a Google sheet, edit it and send feedback to the mBlock 5 stage; and all without the sheet being visibly opened (but this was with me already signed-in to Google!).

The *'Video Sensing'* extension senses motion with an attached video camera. mBlock 5 seems to recognise any web-cam attached to a PC. It does work OK, but it seems to be of rather limited value.

The *'Text to Speech'* extension works well. It makes your projects talk and you can choose from a range of voices. I'm not sure as to its real value though, since this very much depends on the nature of the project in hand; and to work well this needs the words to be written *phonetically* (so that they sound right) and not as conventionally spelt.

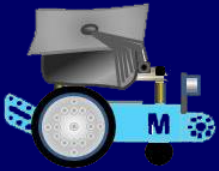
I tested this by using it to speak a sequence of the *'Climate Data'* extension variables mentioned on the previous page. Some climate data output required repeated tweaking before it sounded OK.

The *'Translate'* extension works very well. It will translate text into many languages. I tested this by using *"The pen of my aunt is in the bureau of my uncle"* which it immediately returned in French as *"La plume de ma tante est dans le bureau de mon oncle"*.

The *'Makey Makey'* extension (make-key, get-it!) literally allows you to make a simple key code to operate a hat block. Fun for kids, this is a 'hat' block that requires a sequence of key presses to activate the script to which it is attached (e.g. *'up, right, down, left'*) a fun idea, but essentially useless - why use it when you can just click the 'hat' block to activate the script, unless you can cunningly hide the *'Makey Makey'* script - perhaps hidden behind a large floating comments call-out box.

The *'AI Service'* extension (from Baidu, Inc.) was visible in the extensions centre earlier in the year. Now, at the time of writing, this it has only just been activated for use in beta format - it says that it is currently only available in China (& much of its feedback only appears in Chinese characters!). I haven't therefore experiment with it yet, but it looks both promising and exciting. Opening this extension actually makes five new categories available: *'Speech Recognition'*, *'Text Recognition'*, *'Image Recognition'*, *'Human Body Recognition'* and *'Natural Language Processing'*. - Wow!

N.B. Beware, the only way that any *'Extensions'* that you have added to the *'Blocks'* categories panel can be removed is by reopening the *'Extension center'* and then clicking the *'Delete'* button below each individual *'Extension'* pack. If you wish to clean up mBlock 5's available block categories panel by deleting more extensions, it becomes a rather tedious task to reopen the *'Extension center'* each time since it closes after each deletion.



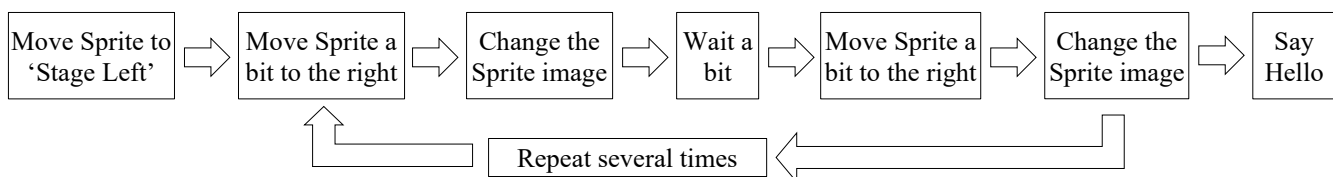
Chapter 11 - Programming with mBlock 5

Despite what I have said earlier about programming using the 'Devices' tab blocks to control mBot, it's no bad thing to do a little programming using the 'Sprites' tab first. This is actually not a bad way to experiment with mBlock 5 - so switch from the 'Devices' tab to the 'Sprites' tab.

The principles are identical, so you can try whatever you want here, and since you are not communicating with mBot, you can't wreck anything, and you only need to save these project files if you really want to.

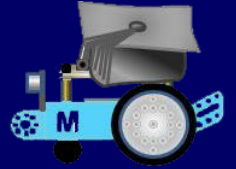
My demonstration exercise shown below gets the default sprite 'Panda' to walk on to the screen and say something. I used this exercise to encourage Emma to understand the programming techniques of Scratch immediately after she had gone through the 'Blockly' based scripting exercises on her tablet.

The thought process (the basic algorithm) for this sequence is as follows:



Open mBlock and choose the 'Sprites' tab (note that the block categories and the blocks themselves change when you switch tabs). Drag the required command blocks from the 'Blocks' area to the 'Scripts' area as shown in the diagram below and slot the blocks together in the order shown on the right of the diagram:

If you make a mistake, you can click the right button of your mouse and choose 'delete' from the drop down menu to zap an errant block.



You can also choose 'duplicate' from the right-click menu too, so try duplicating the purple 'next costume' block after you have dragged the first one across (because you need two of these) - leave the duplicate at the bottom of the 'Script Area' until you need to drag it into place - *duplicating blocks or groups of blocks is very a useful trick to learn*. You can either click the green flag on the top of the screen (below the panda) to let mBlock run your programme or you can click the 'Hat' block at the top of your script. Clicking here is OK - but it is essentially a bad habit - which sadly I still mostly do!

You will also see the green flag below the stage light up when it is clicked, and the red stop button dim - note too that the red stop button lights up again when the code completes and the programme stops.

Try adding a second sprite - anything will do. If you click on the new sprite, you will see no script in the 'Scripts' area but if you click back on the Panda sprite you will see your block script once again. This little experiment will show you that scripts are indeed attached to individual sprites. You can have many sprites in a project, and you can have many scripts (or none at all) attached to a sprite.

Try clicking on the 'Hat' block at the top of your sequence of blocks, hold down your mouse key and drag your blocks left from the scripts area towards the two sprite icons showing in the sprites pane on the left of the screen. When your cursor is over your newest sprite, the sprite icon will gain a blue background and 'wobble' a bit from side to side, let go of the mouse button and you will see that you have duplicated your set of scripting blocks and now both sprites have the same programme attached to them - try moving your new sprite by running the programme - you could also try modifying it so that it is slightly different to the script attached to 'Panda'.

You have just learned how to duplicate scripts; and moving scripts this way is a very a useful trick to learn.

Programming your mBot robot for the first time

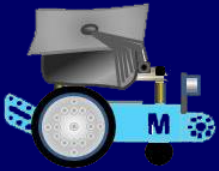
With a new (out-of-the-box) mBot, you have the following DEVICE programming options:

You can write scripts to control each of mBots **outputs**:

- (i) the on-board Buzzer.
- (ii) the on-board LEDs.
- (iii) Motor 1 & Motor 2 (the drive wheels).

You can also write scripts to access each of mBots **inputs**:

- (i) the Light-Sensor.
- (ii) the Distance Sensor.
- (iii) the Line-Tracker.
- (iv) the IR Remote.
- (v) the on-board Button.
- (vi) the Timer.



You can approach your programming journey any way that you like, but it's a good idea to start your mBot scripting adventure with some simple linear sequence scripts programming mBots output devices (using no loops or control structures). I got Emma to create scripts to play a sequence of sounds with the buzzer and flash the LEDs. The simple exercise shown on the next page is based on what she did.

Programming using the 'Devices' tab of mBlock 5 and using the 'Scripts' area on the right side of the screen is easy. The code is 'live' and it can communicate with mBot as soon as it is created. You don't need to upload anything into mBot, which if it is connected to your computer by Serial cable or by Bluetooth or by Wireless will follow the instructions in your programme as soon as you tell it to run using the 'Green-Flag' ('start') button.

You can and should however **'Save' your programmes frequently** so that you can reload them back in to mBlock 5 at any time (so you don't have to create them again). See Chapter 17 re. saving your programmes (pages 152 and 153).

Here are some suggested programming experiments to explore each of mBot's input and output devices:

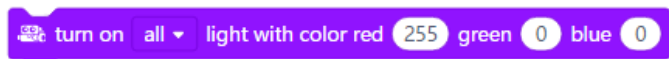
1. Using output from mBots LEDs & Buzzer (light & sound).
2. Using the keyboard to control mBots movements & programming the IR Remote.
3. Using input feedback (distance), (line following) & (light).
4. Using feedback from logic & maths functions.

These will all be demonstrated in subsequent pages:

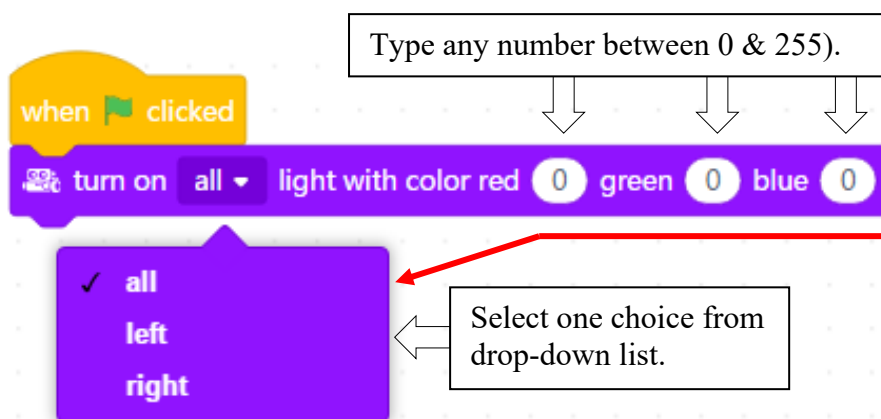
Simple linear scripts to control mBots LEDs & the Buzzer (light & sound):

In this exercise you need to experiment with the following two blocks.

The one that controls mBots on-board LED lights:



And the one that enables mBots on-board Buzzer to play sounds:



To turn *ON* or *OFF* mBot's LEDs, you click on the down-arrow in the blocks left-most window, the one which defaults to 'all'.

As you can see here, it gives you three choices from a drop-down menu and instead of 'all' (both LEDs) you can select to control just the 'left' LED or the 'right' LED.



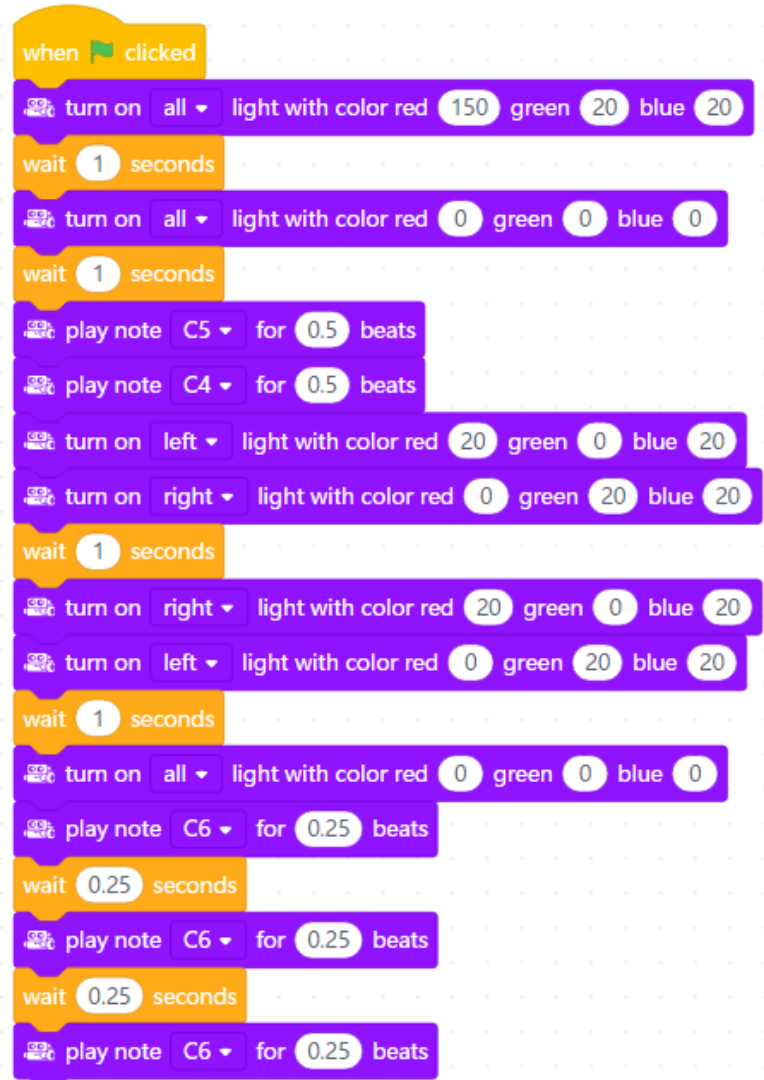
The other three 'bubble' windows in the block shown at the bottom of the previous page require three typed inputs to control the 'colour-mix' of the intensity of the light components (red, green and blue).

You can set any number in these 'bubble' input windows between 0 = 'Off' and 255 = 'Full-On'. If they are all set to zero (0, 0, 0), then the RGB LEDs are set to 'Off'.

N.B. Be aware that many of mBlock 5's programming blocks allow this insertion of your own data - but (as in the 'all', 'left', 'right' choices shown on the previous page) you cannot override the un-editable drop-down lists. Long lists like the tones list in the 'play note' block usually have a slider enabling fast scrolling through the list.

Shown on the right is a fairly typical linear scripting example - an experiment that you can try very early on yourself using, for the first time perhaps, the blocks from the blocks area to control Light and Sound output. You can make up and edit your own version of this sequence very easily.

It is important to learn about (and be confident with changing the data input parts of blocks; either those that have a 'menu' (with a down arrow to select from a list) or the little 'bubble' windows which expect typed input.

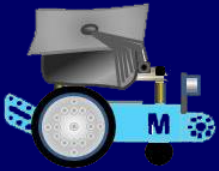


About mBots Buzzer and Recorded Sound Files

Sadly, mBot does not have any capability, in its own right, to play recorded sound files. The only way that mBot can generate sound output is by using the onboard piezo buzzer to play a specific tone.

Since I learned to programme Clive Sinclair's 'Spectrum' in the early 1980's, programming pitch changes for mBots buzzer recently has given me a real sense of déjà vu! To operate the buzzer, you just choose the note that you want in the block 'play note' - or you *can*, if you really want to (and know how to) add numeric values of frequency and rhythm.

The 'Audio Player' extension in mBlock 5 *can* play audio files, but only through Makeblock's own *Me Audio Player* (an acousto-optic output device with a built-in voice decoding chip). Sadly, this seems hard to source; and apart from being included in a newly released (and expensive) 'Talkative Pet' add-on pack is not available as a stand-alone item in the UK.



mBot and Me a Beginner's Guide

Rather than specifying frequency it is usual to set notes with normal musical notation which you choose from the first drop-down menu window of the 'play note' block. The notes are written as C, D, E, F, G, A and B, followed by a number, 2 to 8, which specifies the number (in scientific pitch notation - low to high) of the available octave that you wish to use (there are actually seven and a bit of these - the same number of octaves as a standard piano).

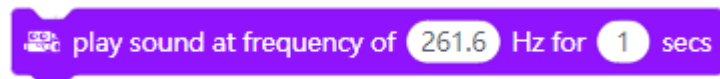
N.B. **Middle C** is the fourth C key from the left on a standard 88-key piano keyboard. It designated **C4** in scientific pitch notation (however in *MIDI*, Middle C is note number **60**).

You can also establish the duration (beat or rhythm) of the note in the second 'bubble' window, but you cannot programme sustained notes (*the period of time during which the sound remains audible and overlapping with other notes*).

You just click in the 'bubble' window and type the duration of the sound you require e.g. enter 1 and the beat will last for one second.



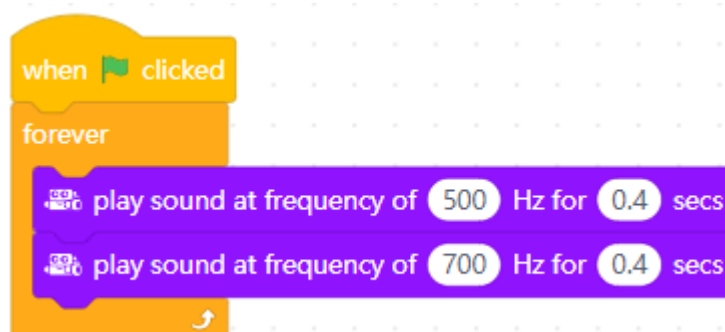
Alternatively if you set the tone within this block as a sound frequency you just type in your own number (in Hz). The frequency of the note **C4** (*middle C*) is 261.625565 hertz. So, if you click in the note frequency window of this scripting block (as shown above right) and type into it 261.6 you will get mBot to also generate (**C4**) (*middle C*).

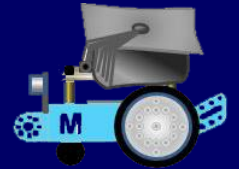


You need to be aware that if you write scripts to output sounds from mBot (such as playing a long note or a short tune) then this will *always* stop the execution of further scripting code until the note or tune completes. The *mCore* board will execute the task in a such a way that nothing else happens until it the tone finishes playing, and this is known in computing as a '*blocking task*' where your script is blocked waiting for the task to complete. This will also disrupt any feedback polling loops in your scripts so if you have, for example, programmed mBot to play a sequence of notes in response to being triggered by a specific input, then mBot will not be able to look or '*poll*' to see if a button has been pressed or what the latest value returned from the line follower etc. may be until the '*blocking task*' of sound generation ends. A glitch like this may cause mBot to go out of control.

I'm not a musician, so I'll leave this section here, but you *might* want to experiment more.

Emma liked the following 'Two-Tone Horn' sound effect, simple but fun - you may want to try it too:





Controlling mBot with Your Computer Keyboard:

To be able to use robotics programming blocks, you *MUST* be on the 'Devices' tab and here you will be able to create a simple set of programmes in the 'Scripts' area that will allow you to control your robot yourself for the first time. You only need to make one script to start with by using the 'Events' block 'when (space) key' pressed and then adding to it, the 'Action' block 'move forward at (50)%'. Edit the hat block by selecting the 'up-arrow' choice in its drop down list of options and then edit the stack block bubble to set 40% power. Duplicate this script four more times and position them in the 'Scripts area as shown below:



Edit each script in turn until all five match the scripts above. The 'Cap' block 'stop (all)' is a 'Control' block. Remember as always to CONNECT mBot to mBlock 5. You will now find that you can use the cursor keys on your keyboard to control mBot - but be prepared to hit the space bar to stop!

For the first time you can make mBot move or stop with a control programme that you have written.

Save this project into your 'My Projects' filing system.

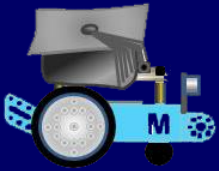
For Info.: In [mBlock 3](#) there used to be both the 'when (space) key pressed' hat block and it's opposite, a 'when (space) key released' hat block.



These were so good to use together in control script sequences like the one that you have just created, since when a key was released it could activate an action like 'stop moving'. Sadly, this block has been omitted from the current versions of both Scratch 3 and mBlock 5.

A rather more advanced version of your keyboard mBot controller is shown on the next page. You might want to attempt this too. Don't panic, it's not that complex but will develop some further skills.

It essentially requires two things, a 'Variable' called 'POWER' needs to be created and the 'Makers Platform' extension needs to be loaded so that you have access to the 'DC motor' blocks (as demonstrated on the next page).



mBot and Me a Beginner's Guide

Click on the 'Variables' button in the block categories and then click on the 'Make a Variable' button. A little window will open asking you to type in a 'Variable' name so type in 'POWER' and leave the radio-button set to 'For all sprites' and then click the 'OK' button. In the 'Variables' menu, under the 'Make a Variable' button you will now see the name of your variable created as a 'Reporter' block together with four more new blocks 'set (variable name) to ()', 'change (variable name) by ()', 'show variable (variable name)' and 'hide variable (variable name)'.

If you later make more variables, then they will also be shown in a vertical list under the 'Make a Variable' button and their names can be chosen in the stack blocks described above by clicking on the little 'drop-down list' arrow next to the existing variable name in the block. Remember to make sure that you are on the 'Devices' tab and that you have added the 'Makers Platform' extension and then create all of the scripts shown below:

```
when clicked
  set POWER to 50
  stop all

when space key pressed
  LED all shows color red
  stop moving
  wait 1 seconds
  LED all turn on light with color red 0 green 0 blue 0
  stop all

when up arrow key pressed
  LED all turn on light with color red 0 green 0 blue 0
  DC motor motor port1 anticlockwise rotates at power POWER %
  DC motor motor port2 clockwise rotates at power POWER %
  LED left shows color green

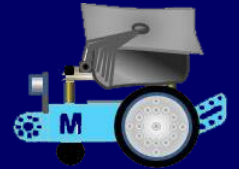
when left arrow key pressed
  LED all turn on light with color red 0 green 0 blue 0
  DC motor motor port1 clockwise rotates at power POWER %
  DC motor motor port2 clockwise rotates at power POWER %
  LED left shows color blue

when right arrow key pressed
  LED all turn on light with color red 0 green 0 blue 0
  DC motor motor port1 anticlockwise rotates at power POWER %
  DC motor motor port2 anticlockwise rotates at power POWER %
  LED right shows color blue

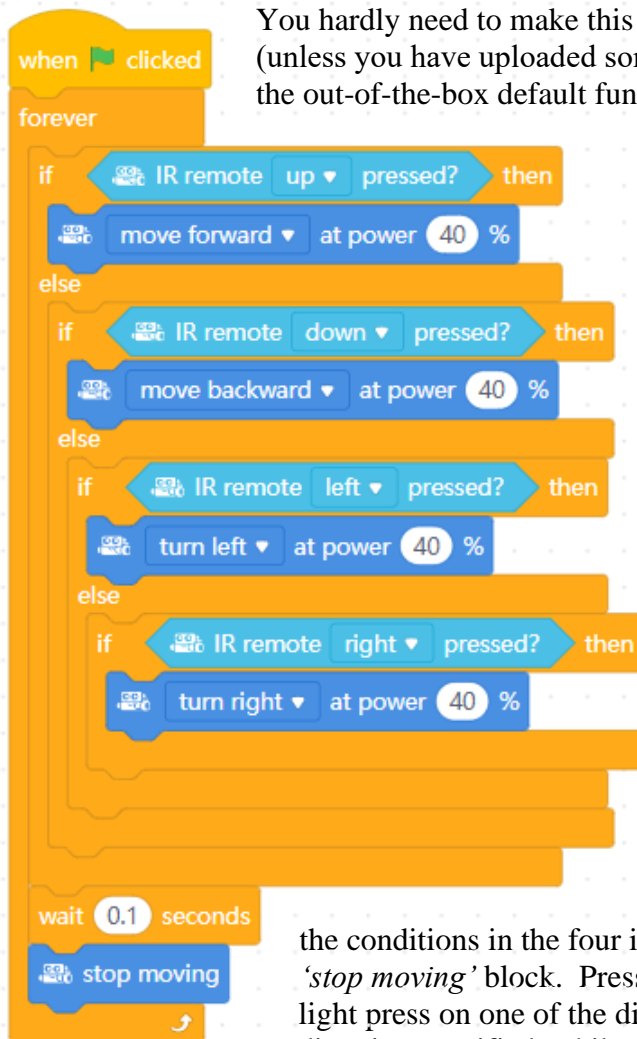
when down arrow key pressed
  LED all shows color blue
  DC motor motor port1 clockwise rotates at power POWER %
  DC motor motor port2 anticlockwise rotates at power POWER %
```

You should be able to see how they work since they are very similar to the simple keyboard control scripts that you have just experimented with and which are shown on the previous page. You can set the speed for the 'DC motor' blocks by setting a new value in your 'Green Flag' initiated 'POWER' variable.

N.B. 50% is good, 35% is about the lowest that you can go without stalling and 100% is naturally the fastest speed. You should also be able to understand the sequence of lights added to each controller script. Note too that there is no block to turn LED lights off (but why not?) so to do this all input values need to be set to zero (0, 0, 0) = 'Off'.



Creating a Control Programme for the Infra-Red Remote



You hardly need to make this script since your little IR remote already works (unless you have uploaded something into mBot's flash memory and no longer have the out-of-the-box default functions loaded!). But it's not a bad idea to go through this short exercise to try to understand how keys on the IR controller *can* be programmed to pass commands - *any* commands to mBot.

The 'Devices' script (shown on the left) looks complex but it just needs to make four decisions - which key has been pressed and what to do next (*and if it's none of them then stop mBot moving*).

There are three 'if/then/else' conditional 'branching' (decision) blocks nested inside each other - just make the first one and then duplicate it twice more, adding each duplicate block sequence into the 'else' part of the one above, editing the keypress directions as you do so. Since there is now only the need for a final (fourth) decision to be made then it does not need an 'else' decision; so this time you can use the slightly simpler 'if/then' decision block.

At the bottom (but inside the opening forever loop) is the final choice that needs to be made if none of

the conditions in the four individual decision making loops have been met - the 'stop moving' block. Press the 'Green Flag' button to activate the script. A light press on one of the direction keys on the IR remote will pulse mBot in the direction specified, whilst a long press will maintain continuous movement in

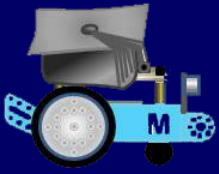
that direction. Releasing the key will stop mBot.

Try it - however, be aware that you can only programme these four direction keys when you have the 'Devices' tab selected; and only when you are in 'Online' mode.

Whilst experimenting with this, I particularly wanted to get the 'COG' key in the centre of the four direction arrows (which incidentally in the 'Boolean' block menu is called 'Setting') to work. It looked so useful as a potential 'stop-all' key, but full programming of the IR key controls only works if your control scripts are **uploaded** into mBot's flash memory to replace its default settings.

This is a little too advanced for now, but the programming codes shown below are those returned when the corresponding buttons are pressed on the IR remote control but these key codes only work if any script using them is uploaded into mBot and run in 'Offline' mode.

A	B	C	D	E	F	^	v	COG	<	>	0	1	2	3	4	5	6	7	8	9
69	70	71	68	67	13	64	25	21	7	9	22	12	24	94	8	28	90	66	82	74



About mBots senses, its Eyes & Ears; and using input feedback (*distance*), (*line following*) & (*light*):

mBot has sensors that it demonstrated for you when you operated it with the Infra-Red remote control unit (which incidentally has a line-of-sight range of approximate 10M). The IR sensor on the front of the mCore board is the first of mBot's sensors receiving signals from the IR remote enabling you to control mBot manually.

The second sensor, also on the mCore board, is a light sensor (which looks like a clear glass bulb sticking straight up out of the board). It is also located at the front of mCore between the two LEDs (note - if these LEDs are activated and set to their brightest white light then this may affect the readings from the light sensor by a minimal increase of perhaps just one or two). The light sensor feedback values are theoretically in the range of 0 to 1000, however I have seen it reporting values of 1004 or 1005.

Sensors are used to detect events or changes in the environment and send information to the electronic components of other devices. While a programme is running it is often required to collect real-time sensor values to help understand the environment e.g. light, sound and distance values.

Data values that may change during the execution of a programme are known as '*Variables*', whilst data values that stay the same during the execution of a programme are known as '*Constants*'. You need something called a '*Variable*' to store and display values from sensors. Giving variables sensible names helps when polling information from them via your scripts (and your scripts become much more understandable too).

If you create a script with a '*Variable*' set inside a '*Forever*' loop you can see real-time feedback values from sensors such as the on-board light-sensor - try moving your hand over the top of the mCore board to see the value fall. If the ambient light is not stable (e.g. from a fluorescent lamp source) you cannot see the rapid change of sensor values very easily, so a '*wait*' block can be added to a simple script to reduce the speed of change in the variable's value which you will then be able to see clearly.

I have not yet really found a use for light-sensor feedback although I guess that it could be used to turn on LED lights as darkness falls (very much like the common occurrences of 'security-lights' around a house or garden). A control programme can store variable values e.g. the range of expected data (highest and lowest acceptable levels and what action to take if they're exceeded). A continuous (forever) process of such data analysis is called a feedback cycle.

Sensors are used to measure physical quantities such as distance, temperature, light, pressure, sound, and humidity. They send signals to the processor on the mCore board to interact with programmes that you can write to make control decisions.

Here are some other real-world examples of sensors in use:

A security alarm system, which may have an infrared sensor which sends a signal when the beam is broken. A heat-sensitive sensor in the corner of a room may detect the presence of a person (*by using mBot's ultrasonic sensor you can try something similar to both of these*). Temperature sensors can be used to control the heating in a building. Magnetic sensors can be used to detect metal and can be placed in roads to monitor traffic flow.



Data transmitted from sensors (such as pressure, light, sound and temperature) is known as analogue data. However, computers can only work with digital data. An interface or analogue to digital converter (ADC) is needed to convert the analogue data from the sensors into digital data that the computer can process and mBot has one of these too.

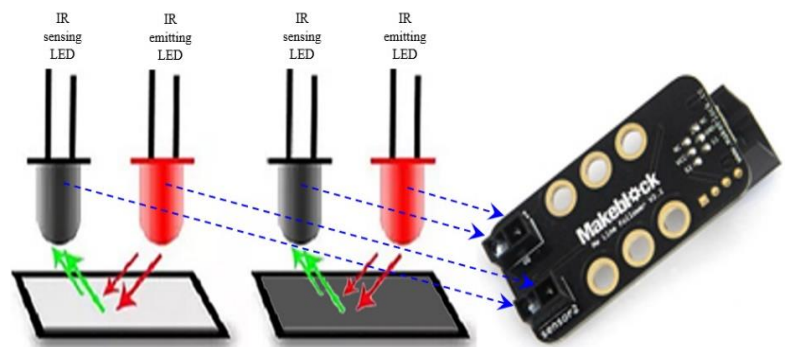
After the on-board sensors mentioned on the previous page (and more importantly perhaps) mBot has two 'add-on' input sensor modules as part of its default components: one to sense distance using ultrasonic feedback to let mBot avoid obstacles; and the other to sense the contrast between a light and a dark surface for its line-following mode.



The Me Ultrasonic Sensor Module on the front of mBot is stylised to look like two 'eyes' when it is mounted in the default position above the 'mouth' shape cut into the front of mBots chassis; however, these are really 'ears' not 'eyes' and they enable mBot to see and recognize objects, avoid obstacles, measure distances, and detect movement using the same scientific principle as bats. The ultrasonic sensor which has a max. range of 4M, measures distance (in cm) by calculating the time it takes for a sound wave to hit an object and come back - it's an echo-sounder!

The Me Line-Following Sensor Module is also mounted at the front of mBot (below its 'mouth') and points downwards; there is quite a bit of science happening here for mBot to follow variations in the contrast of reflected light.

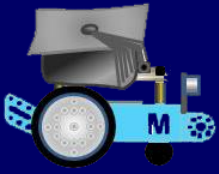
This input sensor module contains both an infrared emitting LED and an infrared sensing LED and uses these to check the contrast between a light and dark surface and mBot comes with a figure-of-eight paper track (for line-following purposes) that you can use.. You can add *some* Me Sensor Modules as needed for your own projects by buying mBot add-on packs.



These are the available (mBot compatible) Makeblock Me Series Modules:

- | | | |
|----------------------|----------------------|-------------------------|
| Me 4 Button | Me Infrared Receiver | Me Shutter |
| Me 7-Segment Display | Me Joystick | Me Sound Sensor |
| Me Bluetooth | Me Light Sensor | Me Stepper Motor |
| Me Buzzer | Me Limit Switch | Me Temperature |
| Me Compass | Me Line Follower *** | Me TFT |
| Me DC Motor | Me PIR Motion Sensor | Me Touch Sensor |
| Me Flame Sensor | Me Potentiometer | Me Ultrasonic Sensor*** |
| Me Gas Sensor | Me RGB Led | Me USB Host |
| Me Gyro | Me RJ25 Adapter | Me Wi-Fi |
| Me Humiture Sensor | Me Serial | |
| Me I2C Scan | Me Servo | |

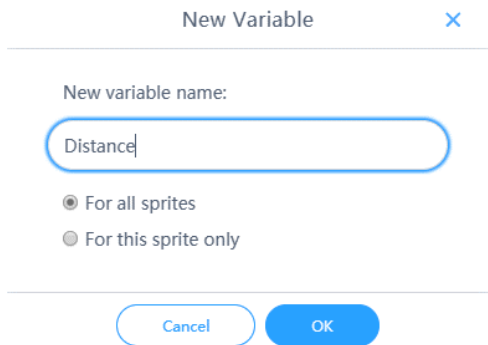
*** the default components supplied with mBot



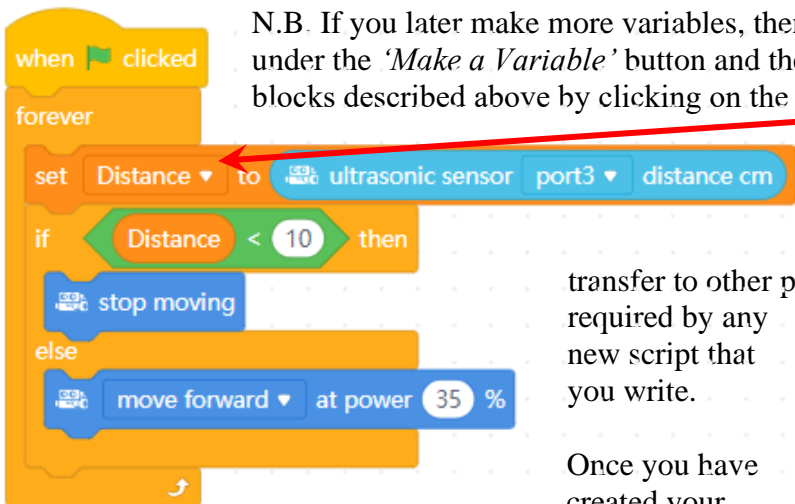
Simple use of the Ultrasonic Sensor

The aim of the simple exercise outlined below is to get mBot to move forwards and stop at a predetermined fixed distance from an obstacle that it detects with its ultrasonic sensor and when / if the obstacle is removed, it continues to run forwards. The ultrasonic sensor module mounted on the front of mBots chassis has two protruding tubes that look a bit like 'eyes'. One of these 'eyes' transmits a high frequency wave that bounces off any intervening surface or object and the wave returns to be received the other 'eye'. This type of sensing is known as echo-location (so they are actually 'ears'!). The sensor is set by default to measure distances in centimetres.

To undertake this exercise, you first need to create a specifically named variable, so click on the 'Variables' button in the block categories and then click on the 'Make a Variable' button. A little window will open asking you to type in a 'Variable' name so type in 'Distance' and leave the radio-button set to 'For all sprites' and then click the 'OK' button. See the illustration on the right).



In the 'Variables' menu, under the 'Make a Variable' button you will now see the name of your variable created as a 'Reporter' block together with four more new blocks 'set (variable name) to ()', 'change (variable name) by ()', 'show variable (variable name)' and 'hide variable (variable name)'.



N.B. If you later make more variables, then they will also be shown in a vertical list under the 'Make a Variable' button and their names can be chosen in the four stack blocks described above by clicking on the little 'drop-down list' arrow as shown in the script on the left.

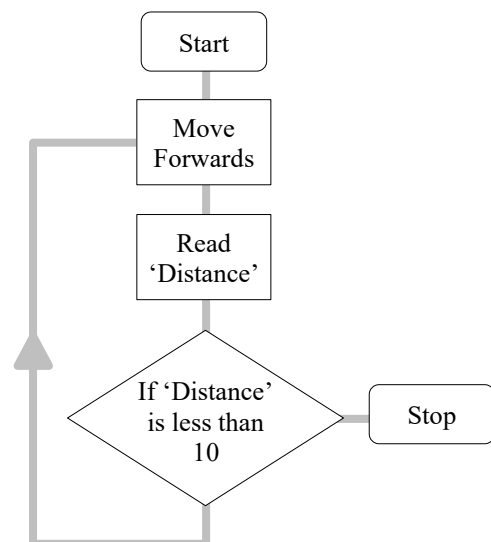
Be aware that any Variables that you create do not remain in mBlock or transfer to other projects, so you must create fresh ones as required by any new script that you write.

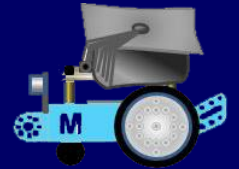
Once you have created your

'Distance' variable, then the 'set - 'Distance' to ()' block becomes available and you can begin to make the 'Devices' script shown in the diagram above.

By using the conditional 'branching' block 'if / then / else' from the 'Control' blocks group you can make your programme branch to do either one thing or the other.

The logic of this sequence is fairly easy to follow without an algorithm, but here on the right: I've shown it anyway.





If you click on the 'Green Flag' (or the 'Hat' block at the top of the script) mBot should run forward until it is less than (or equal) to 10 centimetres away from an obstacle - try using your foot! If the obstacle is removed, it will continue to run forward - *but it will not stop* - something that we will solve next! Make sure that you save your project with a suitable filename.

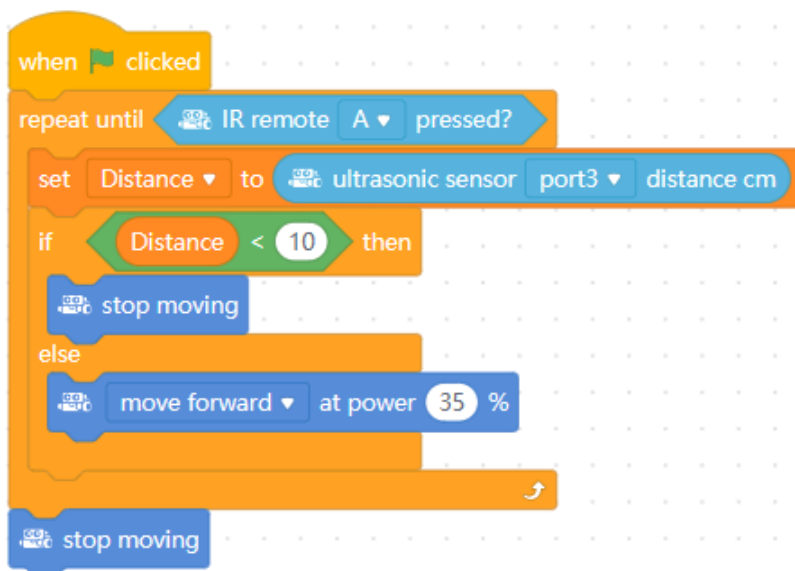
The script now needs to be improved by changing the method of 'looping'; so you need to modify it to look like the script shown at the bottom of the page. This will allow you to press key 'A' (Manual Mode) on your IR remote control to exit the programming loop and stop mBot whenever you want to.

I also suggest that you practice your block nesting skills here too. You don't have to do this bit, but you could modify the 'set - Distance - to ()' stack block in your original script to measure in millimetres (using my method for doing this as described in detail in Chapter 9 (page 41) and shown again below:



mBlock 3 had a very useful 'key (space) pressed / released?' hexagonal 'Boolean' block enabling any specified key on your keyboard to action / exit from decision making scripts. In mBlock 5 this has now been replaced by two 'Boolean' blocks 'when onboard button (pressed / released)' and 'IR remote (A) pressed'; neither of which are, I believe, quite as useful!

Start to modify your simple sensing script by removing the looping 'Control' block 'forever' and replacing it with a 'repeat until' looping 'C' block - this will also loop forever; but only until something that you have specified becomes true (like specifying pressing the 'A' key on the IR remote).



From the 'Sensing' blocks group drag the blue 'IR remote (A) pressed' block into the empty hexagonal slot at the top of the 'repeat until' block. Inside the 'repeat until' block should be your original sequence of 'if / then / else' blocks that you put inside the 'forever' block in your first programme (they need to do the same job).

Finally, you need to add the 'stop moving' block at the bottom (outside the 'repeat until' loop. This will then only be activated when the 'repeat until' bit at the top of the loop is true. If the 'A' key on your IR remote is pressed then

the programme jumps out of its repeating loop and mBot stops.

This modified script is not perfect, but it is indeed a much better solution to the original problem, and you should have learned a little more from this experiment; so save your modified project once again - but this time with a new name.



Another (slightly more advanced) Control Programme for mBot - the Line-Following Sensor:

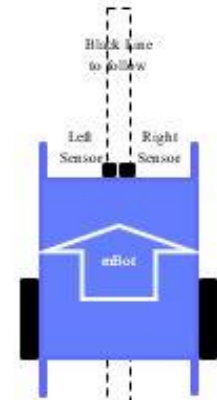
mBot already has something similar to the programme you are about to create built into its memory; but the aim of this exercise is to understand how mBot can follow the black line of the figure-of-eight paper track supplied. When the light energy from the emitting LEDs in the line-following sensor reflect off a surface, information about the reflective properties of that surface is transmitted back to mBot and the lighter the colour of the surface, the more light is reflected (in comparison to the amount of light that is reflected from a dark coloured surface).

The line-following sensor module mounted on the bottom of mBots chassis at the front has two infrared emitting LED / infrared sensing LED pairs on the underside of the module; each pair is mounted about 12mm apart. A pair of small blue indicator lights are visible on the top of the module and can be seen just in front of mBots 'mouth'.

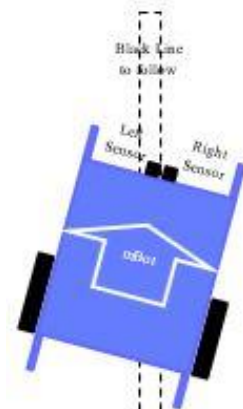
Each blue light is a good indicator of what each LED pair can 'see'. If **both LEDs are ON**, they indicate that there are high levels of reflected light and are over **a light surface** - the sensor module returns a value of '3'. If **both are OFF** this indicates low levels of reflected light (they are over **a dark surface**) and the sensor module returns a value of '0'.

The sensor uses its LEDs to constantly check the contrast between the surface below each of them - if the left-hand blue-indicator light on top of the module is OFF it shows that low reflected light (**a black surface**) is detected **on the left** pair of LEDs and the module will return the number '2'.

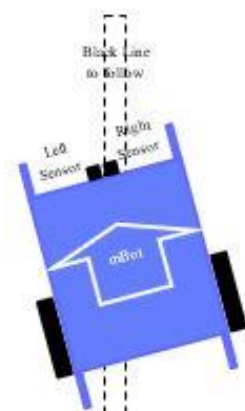
Conversely, if the right-hand blue-indicator light on top of the module is OFF it shows that low reflected light (**a black surface**) is detected **on the right** pair of LEDs and the module will return the number '1'.



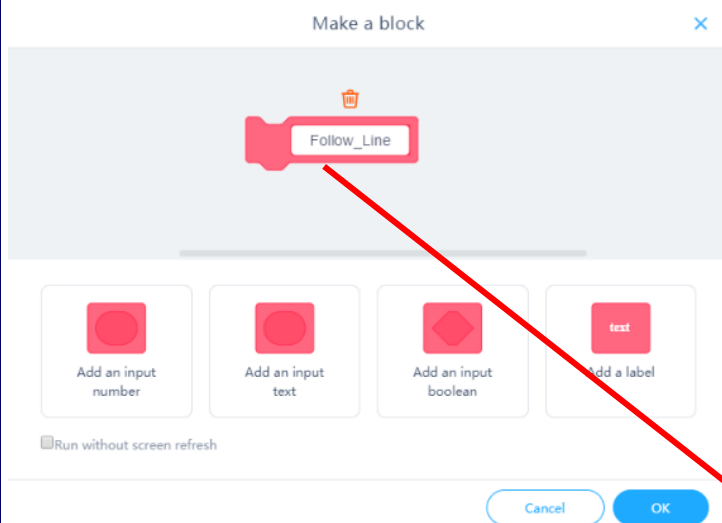
mBot moves **FORWARDS** as it senses black (or low reflected light) and returns a value of 0.



mBot needs to turn **LEFT** as the right LED pair senses white (or high reflected light) and returns a value of 1.



mBot needs to turn **RIGHT** as the left LED pair senses white (or high reflected light) and returns a value of 2.



To complete this exercise, you need to create a variable called 'Line_Sensor_Value' which will hold one of four values (0, 1, 2 or 3) which are returned by the sensor.

You also need to create, a custom 'My Blocks' block called 'Follow_Line'.

This will define the new stack block shown here on the right:





This 'Devices' tab exercise does introduce you to defining a 'My Blocks' block for the first time and the script for your self-defined 'Follow_Line' block is shown below.

```

define Follow_Line
set Line_Sensor_Value to line follower sensor port2 value
if Line_Sensor_Value = 0 then
  move forward at power 35 %
if Line_Sensor_Value = 1 then
  turn left at power 35 %
if Line_Sensor_Value = 2 then
  turn right at power 35 %
if Line_Sensor_Value = 3 then
  move backward at power 35 %
  
```

As you can see, the 'Follow_Line' block has four simple 'If' decisions - if the first is not true then the next is tried and so on. Since the 'Follow_Line' block is inside a forever loop then this is a constant circle of reading the sensor feedback value and reacting in response to the numeric value returned. You will also need to make two separate, simple and very short 'start' and 'stop' scripts too (see below):

```

when up arrow key pressed
  forever
    Follow_Line
when down arrow key pressed
  stop moving
  stop all
  
```

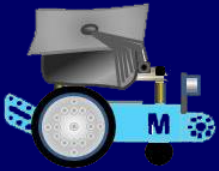
N B. The code blocks (shown above left underneath the 'Follow_Line' hat block) *could all* be inside the forever loop of the primary 'when (up arrow) key pressed' 'start' script (instead of the 'Follow_Line' block shown) but using a self-defined block, as you can see, is a much neater way to do it. Note too that it is often not a bad idea to have several shorter, isolated (and easier to test) sequences of blocks in the mBlock 5 'Scripts' area. Using 'My Blocks' is a good way therefore of achieving this.

Save your file with a sensible filename as usual.

Programming mBot's on-board Light Sensor:

Sensors are used to detect events or changes in the environment and send information to programmes or directly to other electronics devices. When the ambient light around mBot's on-board light sensor changes, then, if you create a suitably named variable and make it visible on the mBlock 5 stage it will show feedback of real-time readings.

This exercise will show how to capture light sensor values and display them as real-time feedback. Other sensors usually use the same method to collect and display data too.



mBot and Me a Beginner's Guide

The on-board light-sensor returns values in the range of 0 to 1000. If mBot is exposed under sunshine then it probably returns values > 500, in evening light, perhaps only 0 to 100 and indoors (under artificial lighting) perhaps values between 100 to 500.

You can test this for yourself - so try experimenting with the following 'Devices' tab script. You will need to create a variable called 'Light_Level' to display feedback and the 'Forever' block will ensure that the light sensor is constantly updated to display real-time values.

```

when clicked
  forever
    set Light_Level to light sensor on-board light intensity
    if light sensor on-board light intensity > 900 then
      play note C5 for 0.1 beats
  
```

If the ambient light is not stable (e.g. from a fluorescent light), then you will see a rapid change of sensor values, so if necessary add a 'wait' block anywhere inside the 'Forever' loop (set to about 0.5 secs) which will reduce the speed of the value changing in the variable and you will be able to see the feedback values rather more clearly.

Click the 'Green Flag' or the 'Hat' block to start the sequence which will run within the 'Forever' loop until the hat block is clicked again or until the 'Stop' button is clicked. The

'Light_Level' variable on the mBlock 5 stage will show a constantly changing figure of approximately one thousand and the on-board Buzzer will beep repeatedly.

Try moving one of your hands close to the top of mBot's mCore board. The 'Light_Level' variable will drop (into the mid-300s). If the light level drops below 900 the on-board Buzzer will stop playing a tone. Raise your hand a little and the 'Light_Level' value will rise, and the note will resume playing when the value rises over 900.

You could try other simple experiments in the same vein too.

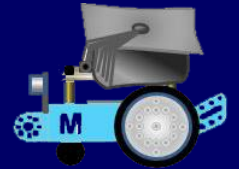
You can for instance easily get mBot to react to the light level in a room. The sample script on the right makes mBot move and creates a soothing (pinkish) LED light mixture when it begins to get dark.

In this case 'dark' is any value less than 100. To test this script, you may want to temporarily increase that value to 600 or 700.

Adding more 'movement' blocks and LED output sequences inside the 'If' section could perhaps create a 'Dancing mBot' sequence!

```

when clicked
  forever
    set Light_Level to light sensor on-board light intensity
    if light sensor on-board light intensity < 100 then
      move forward at power 50 %
      turn on all light with color red 200 green 50 blue 150
    else
      stop moving
      turn on all light with color red 0 green 0 blue 0
  
```



Programming using Feedback from Logic & Maths Functions:

I found that when I began messing about using simple scripts with Emma that the following two exercises were a great success - so much so, that we returned to them several times to create new and improved versions. Look out for them again when we come to programming the add-on LED panel (see Appendix 1 on page 184).

N.B. Scripts like those shown below (with no robotics commands) can be written either on the 'Devices' tab or on the 'Sprites' tab, since both will return their feedback to variable monitors on the stage.

Random Numbers 1 - throwing a 'dice'

For this you need a variable called '*Random_Number*' to hold and display a random number and I added two more variables called '*Text_1*' and '*Text_2*' to hold some extra feedback in the form of text messages displayed on the stage - a much neater way of providing textual feedback than getting a sprite to give out messages using the '*say*' or '*think*' blocks from the '*Sprites*' tab '*Looks*' blocks. You could use just the '*Text_2*' feedback Variable if you want to (& the "Hey Emma" bit could have been added into this variable too).

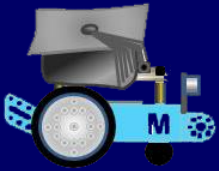
The script and the displayed feedback variables (shown above right) are both very self-explanatory. The script sets a '*Random_Number*' variable between 1 & 6 using the '*pick random*' block from the '*Operators*' block group and then concatenates the result with the second text variable.

My '*set (Text_1)to ()*' stack block just returns whatever text is entered into it. The set 'Text_2' stack block uses a '*join*' block from the '*Operators*' block group to concatenate the text "You have thrown a " with the '*Random*' Variable.

Every time you click the 'Hat' block, a new random number between 1 and 6 will be created and the message will display the new number (although if it generates the same number you will not see a change).

Save your project with a suitable filename.

Setting the variable monitors on the stage to have no labels actually looked much better and Emma liked having a digital 'dice' quite a lot, so we then devised the next experiment to create again using random numbers. This exercise is described on the next page.

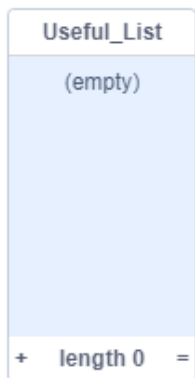


Random Numbers 2 - the 'Rock / Paper / Scissors' game

This exercise uses the same variables as those shown in the previous exercise but with the addition of something new - a 'List'. To make a list, click to select the 'Variables' block group and then click on the 'Make a List' button.

A little window will open (just like the one for making Variables) asking you to type in a name for the list - call it 'Useful_List' (so original !) and click the OK button.

The name that you gave to your list will now become a 'Reporter' block with an assortment of eleven useful blocks below it (see the diagram on the right):



A little window with the name of your list showing at the top will open on mBlock's 'Stage'. The middle of the window will be totally empty - signifying no list contents yet.

Click the little plus (+) button in the bottom left corner and a new blank list entry slot with a '1' in front of it will appear in the window - this is for you to type your first list item - type "Rock" and click anywhere on mBlocks screen to enter the data.

Click the little plus (+) button again and a second slot will appear - enter "Paper" and then create a third slot and enter "Scissors".



That's it, the new list is complete. You can re-size the window by clicking and dragging the bottom right-hand corner and you can click and drag anywhere else on the background to move and reposition the window anywhere on mBlock's 'Stage'. If you click in any of the list slots, you can edit the contents or click the (x) button to delete the slot and you can also right-click and 'hide' the list from view if needed.

The 'length' number at the bottom of the list you will have probably worked out for yourself is the number of entries in the list.

You can use a list like this in a very similar way to a 'VLOOKUP' table in Excel by return anything stored in the second column windows by specifying a number in the first column windows. A specified number returns output from the second column of the row specified by the number.

Reopen your simple 'dice throwing' script (the previous exercise) and save it with a different filename. You can now modify it to look like the one shown at the top of the next page.



```

when space key pressed
  set Random_Number to pick random 1 to 3
  set Text_1 to Oh! Emma
  set Text_2 to item Random_Number of Useful_List
  
```

Random_Number 3

Text_1 Oh! Emma

Text_2 Scissors

You should be able to understand how this works. A random number (1,2 or 3) is generated and the 'Reporter' block 'Random_Number' (let's say it's a 3) is used as the lookup number in the 'List' - row 3 returning 'Scissors'. Save your basic game project with a sensible filename.

Emma thought that pressing the space key as well as using your other hand to make your chosen game shape wasn't very good, so we decided to modify and improve this script a little more by involving mBot.

```

when space key pressed
  forever
    set Distance to round ultrasonic sensor port3 distance cm * 10
    if Distance < 200 then
      set Random_Number to pick random 1 to 3
      set Text_1 to Oh Emma
      set Text_2 to item Random_Number of Useful_List
      wait 2 seconds
      set Random_Number to 
      set Text_1 to 
      set Text_2 to 
  
```

Since this modification involves a robotic block, this script needs to be created on the 'Devices' tab and not on the 'Sprites' tab. You could make the whole thing again from the beginning, but if you open your project file from the exercise above then might have written it on the 'Sprites' tab, and you can't drag scripts across to the 'Devices' tab like you can duplicate scripts by dragging them between sprites.

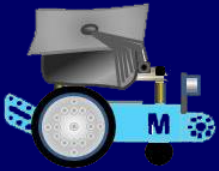
But, your variables and the list that you created are available to both 'Devices' and 'Sprites', so you just need to remake the block script again on the 'Devices' tab (switching briefly to the 'Sprites' tab if necessary to remind you what blocks were used).

You need to add an additional variable called 'Distance' and then modify the script as shown on the right. Make sure that mBot is 'Connected' before you run the script. Nothing happens until mBot detects movement just in front of its sensor. This enabled

Emma to make the 'game hand-shape' and mBot then responded by running the rest of the script to generate its random response. Finally, the script pauses for two seconds and then supplies a blank (empty) response to the three output monitor windows on the stage.

This modification made the game much more realistic to use with mBot as the opponent and Emma really did like this version - but watch out (as with the basic dice-thrower exercises) there are yet more enhanced versions of this to follow as this book progresses!

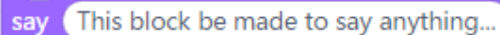
Save your project once more with another (different) filename to the exercise at the top of the page.



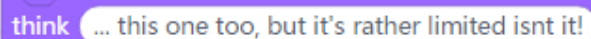
Chapter 12 - mBlock 5 - Graphics Programming

Using textual feedback on the stage is all well and good for the simple levels of 'dice' throwing and the 'Rock, Paper, Scissors' game discussed in the previous chapter. 'Pictures', as they say, 'speak a thousand words' so being able to manipulate graphics on the mBlock 5 stage is an important skill to develop in its own right. Stage graphics can be purely sedentary - just a meaningful backdrop for instance or a sprite image added in front of the backdrop as an 'actor'. Graphics such as mBlock 5's 'Panda' sprite could very easily be programmed to 'say' the outcome of a random number dice throw in a speech bubble.

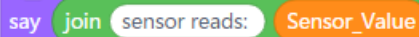
At the level of understanding that we are working at now, it is only possible to display words on the stage by using the two blocks 'say' and 'think' into which



say This block be made to say anything...



think ... this one too, but it's rather limited isnt it!



say join sensor reads: Sensor_Value

you can put any alphanumeric characters (combined with variables data too if you want) into a speech call-out bubble or a thought call-out bubble.

Using these blocks, sprites can display meaningful text and sensor information; but who really wants the default sprite to be saying or thinking feedback from mBots sensors ! ? ! If you create a blank sprite in the costume editor, to use for text output using these blocks then the bubbles rather eerily appear out of thin air. Surprisingly, the 'say' and 'think' blocks can display a lot of text - approx. 480 characters (24 characters per line x 20 lines).

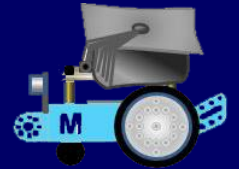
A modification wish: If a new Scratch block could be created, perhaps called 'label' and with a simple, non-specific outline (such as a round-cornered rectangle) then it would be much more useful than the 'say' & 'think' call-out bubbles and this would be *VERY* useful. Creating an additional block such as this might, I guess, be fairly easy for the MIT group (or Makeblock) to programme and add to the 'Looks' blocks collection?

You can if you want to, use the 'text' tool in mBlock 5's costume editor to make labels of any size and shape that can be positioned anywhere on the stage. The fonts are rather limited and there is no justification or emphasis like **Bold** or *Italics*, but you can colour text in any shade you like & you can cheat a bit by duplicating a text sprite, changing its colour and then positioning it over the top of the first to create a drop-shadow effect.

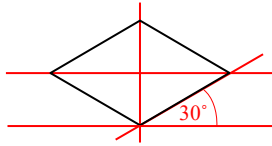
Throwing 'real' dice on to the mBlock 5 stage

To do this you need to create an animation, using a graphic image of a dice showing every possible combination of spots on the dice. You need therefore six matching images showing each of the different spot permutations. You could take photos of a real dice to do this (or source dice-face images from the internet) but it is much better to create your own drawn graphics. (See Chapter 14. This provides a lot of information about how to draw and create sprites). **To use graphics on the stage that you can manipulate via scripts, you need to be on the 'Sprites' tab.**

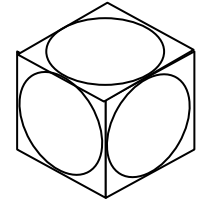
Animations in Scratch programming are achieved by changing a sprite's costume, so the basic programming principle here is if 'Random_Number' = 1 then show costume '1'.



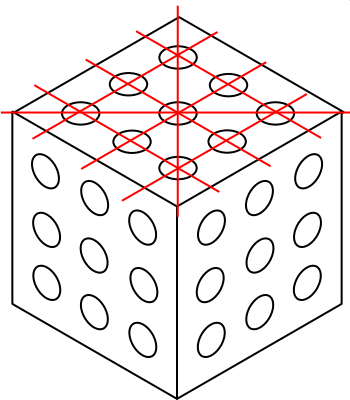
I already happened to have a set of dice images (drawn in 'Word') at least twenty years ago to illustrate random numbers in 'Excel'. Making them is not difficult - but do refer to Chapter 14 for guidance. I based my drawings on 'Isometric Projection' a technical drawing way of showing pseudo 3D images with all horizontal lines being drawn at 30° and all vertical lines remaining at 90°.



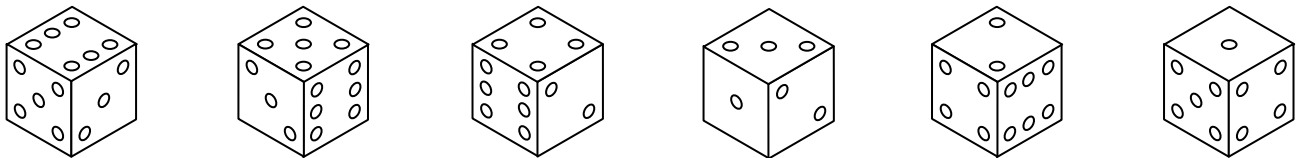
To make the top of a cube, draw this 'diamond' using the 'Freeform Shape' tool. Duplicate it twice and rotate one duplicate by 60° and the other by -60° and position them as shown on the right.



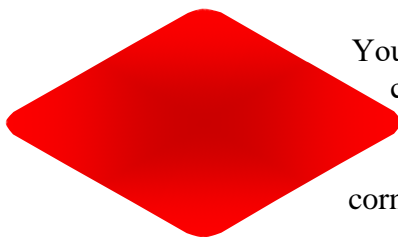
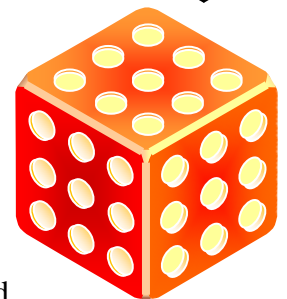
Add an ellipse to fill the top of the resultant 'cube' and duplicate it twice too. Rotate one duplicate by 60° and the other by -60° and position them as shown here. You now have now created some very useful basic isometric rectangles and circles for 3D work.



There are nine positions where a spot can be found on each face of a dice, so divide up the top face of your cube with more 30° (isometric) construction lines as shown in red on the left. Group the set of red lines and rotate them and add them to each face. Shrink your three ellipses to a suitable spot size and then duplicate and position them as shown on the left so that there are nine spots carefully positioned on each of the three visible faces of the cube. Delete the three sets of red temporary construction lines. Group everything that you have drawn so that your nine-spot dice cube becomes one object. Duplicate it five times (so that you have the six dice face permutations that you need and then (only then) delete unwanted spots on each face of each of the cubes to leave them as shown below.



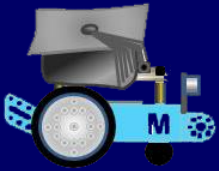
You may wish to colour your dice before you make your five duplicates - I did exactly that, as you can see on the right, making each of the 'diamond' faces a colour of your own choosing and then making the spots a contrasting colour. To get the effect of reflection on the three internal edges of the dice I added a line matching the colour of the spots in each direction and added them to each group.



You don't have to do this bit, but I also created a round cornered 'diamond' shape (shown on the left) for each of the three sides of my cube using a subtly graduated fill and then deleted the original sides. You can see from the diagram (above right) that the rounded corners and edge lines do add a more realistic look to the finished dice.

Once I had completed my six dice drawings in 'Word' I converted them into .png files so that they could be added to my 'Sprites Library' in mBlock 5 (see Chapter 14 on pages 89 to 96 for more on this).

Back in mBlock 5 I could now add the six dice images (named '1' to '6') as individual costumes of a sprite which I named as 'Dice' - once again such an original choice of name!



I decided that I would start afresh with totally new scripts for this project rather than modifying the very simple dice thrower script described on page 67.

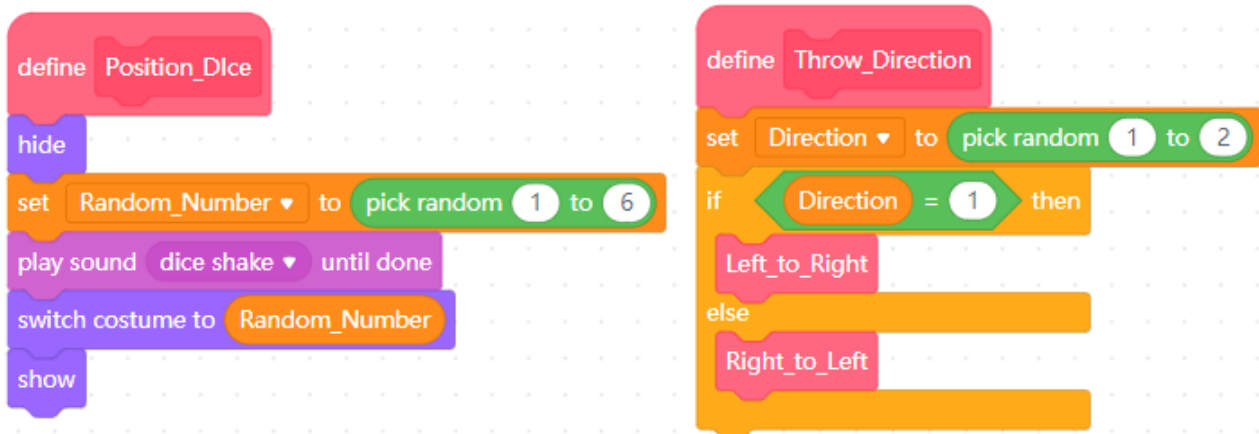
I created a new backdrop for the stage by going into the graphics editor and filling the whole of the 'paint' area with a shade of dark-green (Hue: 30, Saturation: 100 and Brightness: 40) to emulate green baize. My 'Dice' sprite looked good on this.

My concept was to have one dice being rolled from the top of the stage (the starting corner being chosen at random); the sprite starting small and tumbling to its final resting place in the opposite corner at the bottom of the stage with the sprite costume being selected by the random number 'Operators' block we had used earlier. To do this neatly and clearly I decided to use self-defined 'My Blocks' to maintain clarity.

Start by creating the four 'My Blocks' hat blocks shown on the right and then create the primary 'start' script using the 'when (space) key pressed' 'Hat' block shown on the left of the diagram.

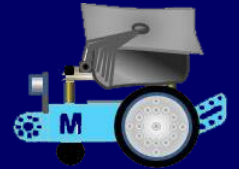


Next you need to populate the four self-defined hat blocks. The top two of these are shown below:



You can see here that the 'Position_Dice' block throws the 'Random_Number' and chooses the costume matching it. You will also notice that I have added a 'play sound' block which you can omit if you wish whilst you are creating this project. The short sound that it calls is a recording that I made of dice being shaken in a soft backgammon dice cup. It's worth the effort, because this does create the illusion that dice are about to appear! The 'Throw_Direction' block uses another random number 'Operators' block to choose from which side of the stage the dice will be 'thrown'.

The 'Throw_Direction' decision then calls one of the two left /right 'dice-animation' script sequences (shown at the top of the next page). They are essentially identical, so make one first and then duplicate the blocks to add to the second. All that you need to do is reverse all references to the x position in your duplicated blocks making positive numbers *negative* and negative numbers *positive*.



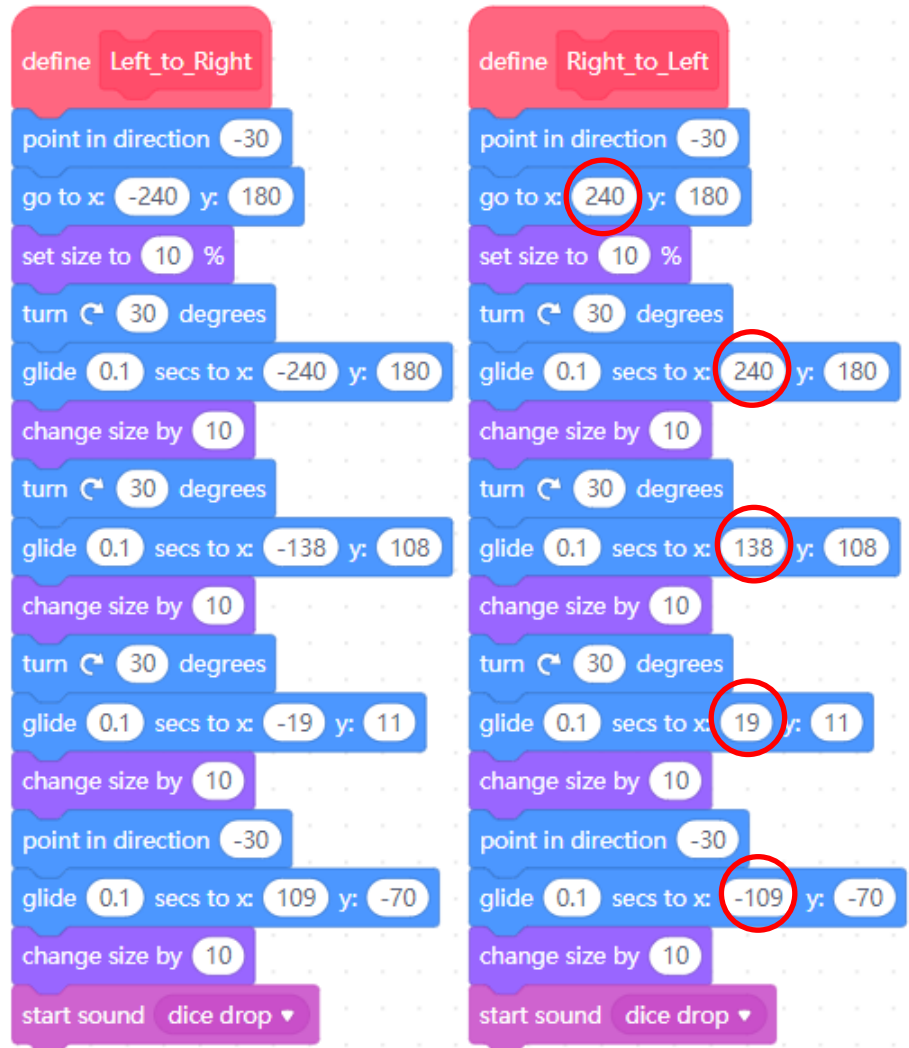
As mentioned on the previous page, the 'Right_to_Left' script is a duplicate of the 'Left_to_Right' script with all five (x:) coordinate references reversed (positive to negative and negative to positive). These are indicated by a circle around them.

The first three blocks in both scripts position the 'Dice' sprite at its starting (top corner) position, rotate it by 30° and shrink it to 10% of its drawn size. The next three blocks increase the sprite size by 10%, turn it again by another 30° and move it smoothly (glide) to a new position on the stage.

This sequence is repeated for another three times until the sprite has arrived in the opposite bottom corner and achieved 50% size.

You may need to adjust these percentages to match your own 'Dice' graphic size. Finally at

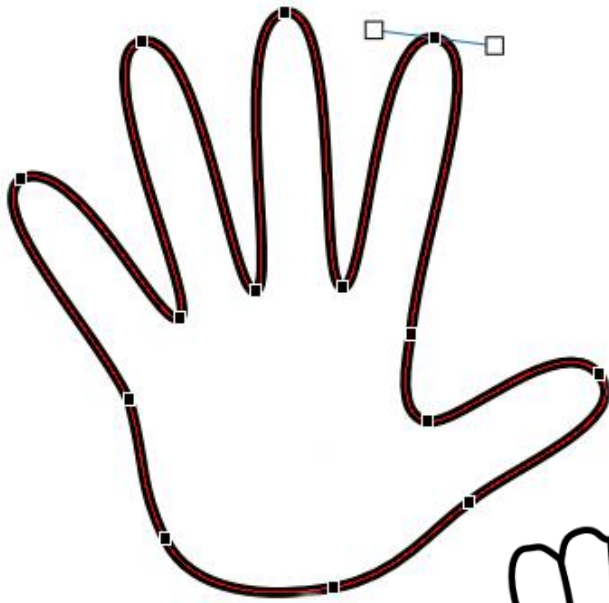
the bottom of each script is another 'play sound' block (which you can also omit if you wish whilst you are creating this project). The short sound that this one calls is another recording I made; this time of dice landing on a backgammon board and as before, this too does complete the illusion that a dice has 'really' been thrown!



Save this project with a suitable filename; and there you have it, real moving graphics on the mBlock 5 stage. Emma was really impressed by this version and although the graphics creation methods were a bit beyond her she did understand the programming including the dice 'tumbling-in-perspective' sequence.

Playing 'realistic' Rock, Paper, Scissors against mBot using the mBlock 5 stage

This is a little harder to achieve than the graphical dice project above. First, you need three images of the required hand shape. You could take photographs of someone's hand for this or get images from the internet (but be careful not to infringe copyright). I went for the option once again of drawing them in 'Word' using the same the 'Freeform Shape' tool to create simple thick outline drawings (using 'Edit Point' nodes to adjust each line). These three Rock, Paper, Scissors drawings are shown at the top of the next page:



The image on the left shows the edit points (or nodes) around one of my three 'hand' drawing.

At the top of the first finger you can see the selected node has two levers which use 'Bezier Curve' attraction to smooth the curve passing through the node. This node is a 'smooth' point which works like a see-saw - if one lever goes up, then the other goes down which affects the path of the line through the node.



Pulling the handle at one end of the lever also operates both levers (either in or out) - the wider the lever handles move apart, the more the line is stretched out either side of the node - try it, it's so easy! Once I had completed my three 'hand' drawings in 'Word' I converted them into .png files so that they could be added to my 'Sprites Library' in mBlock 5

```
when space key pressed
  forever
    set Distance to round ultrasonic sensor port3 distance cm * 10
    if Distance < 200 then
      set Random_Number to pick random 1 to 3
      broadcast message1
      wait 2 seconds
      set Random_Number to 0
```

New block here

I opened my last 'Rock, Paper, Scissors' project file described on pages 68 & 69) to modify the script that I had created on the 'Devices' tab (as shown here on the left).

I no longer had the need to display the 'Text_1' or 'Text_2' variables on the stage so I deleted them from the 'Variables' blocks list. I then removed from the script the corresponding 'Stack' blocks.

I needed to add a new block to replace them (as shown here above). This was an 'Events' block 'broadcast (message1)' - This broadcast block is a **VERY IMPORTANT** addition, but more about this on the next page.

I then switched to the 'Sprites' tab. Remember, this is where you can programme sprites on the stage.



Just like the dice images in the last exercise, I added my three drawings as individual costumes into a new sprite which I called 'Hand'; each costume appropriately being named as 'Rock', 'Paper' and 'Scissors'.

On the 'Scripts' area of the 'Sprites' tab I created the script shown on the right. It is very easy to understand and there are three individual decisions to be made, each choosing the costume to match the random number generated by the script on the 'Devices' tab and stored there in the 'Random_Number' variable.

The last three blocks show the hand-shape costume that has been chosen, wait for two seconds and then hide it again.

This matches the concept of the script on the 'Sprites' tab showing 'nothing' in the 'Random_Number' variable monitor window after 2 seconds. Hiding the graphic (or clearing the monitor window) is a way of mBot saying "Ready! Play again".

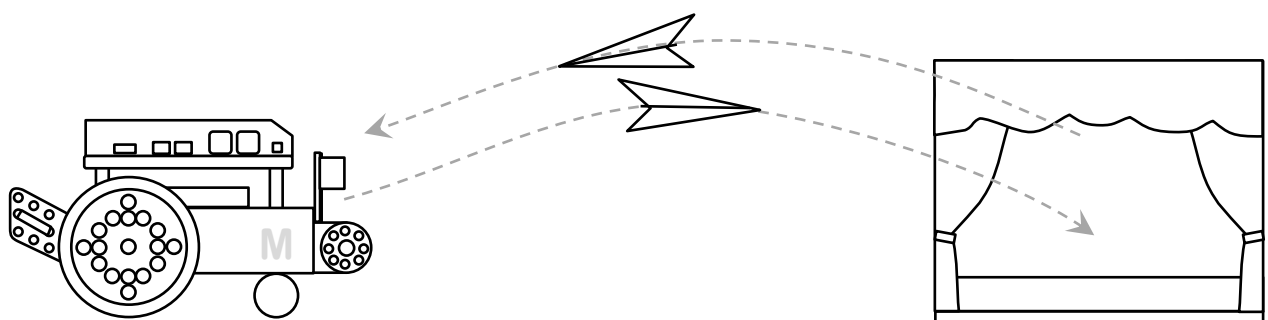
Save your project with a new filename and test it.

Emma now not only had mBot sensing her hand movement in front of the sensor, but mBot responded by showing its own hand-shape choice on the stage; and this looked so good when run in 'Presentation Mode' too!

The clever bit of 'magic' here is the hat block 'when I receive (message1)'. This accepts the signal 'broadcast (message1)' sent from mBot via the script on the 'Devices' tab and that allows the contents of the variable 'Random_Number' to be used by a script on the 'Sprites' tab to choose which graphic is displayed on the 'Stage'.

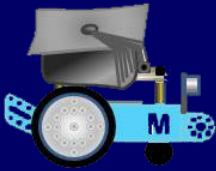
```

when I receive message1
if Random_Number = 1 then
  switch costume to Rock
if Random_Number = 2 then
  switch costume to Paper
if Random_Number = 3 then
  switch costume to Scissors
show
wait 2 seconds
hide
  
```



To summarise, scripts written on mBlock's 'Devices' tab CAN send data as part of a broadcast message to be received by scripts created to receive them on the 'Sprites' tab for use on the stage. This is such an important concept to grasp that I have explained it in some considerable detail in the next chapter.

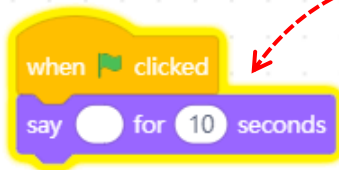
If you have completed the last two graphics programming exercises then you have done very well.



Chapter 13 - Discovering Broadcast Messages

To 'Green Flag' or not to 'Green Flag' - that is the message ...

... In Scratch (and therefore mBlock 5 too) the 'Green Flag' is a programming feature that will, when the icon below the stage is clicked, set a chain of events in motion starting all scripts in a project that begin with the 'when ('Green Flag') clicked' hat block.



You may see that all scripts attached to a sprite thus hatted (and clicked as described above) will be **illuminated**, for the period of time taken to run each script. This shows that they *are* being run and this is useful for instance if you want to populate variables with data at project-start-up. A 'Green Flag' icon click runs all such scripts in a project in parallel; whilst a 'hat' block click only runs the script that is clicked. As mentioned on

page 53, clicking on the 'Hat' block at the top of any script on the stage is something that you may well have got into the bad habit of doing in some of the earlier exercises in this book? - I did & still do!

As your projects get more complex, you must however use the 'when ('Green Flag') clicked' block with some care. My recommendation is not to use more than one 'Green Flag' hat block in a single project since this can create errors that are difficult to diagnose; these errors often only appearing occasionally with every other start of the project and such errors can cause timing issues.

If at all possible, therefore, use just one 'Green Flag' block followed by 'Broadcasts' to activate all other scripts that need to run when the project begins.

When creating a project, it is fully possible to avoid using the 'Green Flag' at all if you design it to be started by a key on the keyboard being pressed or a sprite on the stage being clicked.

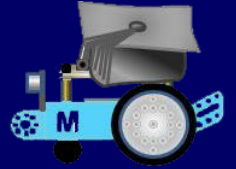
It is possible too to auto-run a project by adding a 'when (timer) > (-1)' block to a project; and then when the project is opened, the 'Green Flag' will be activated and any 'Green Flag' scripts will auto-start as described above.



N.B. If you are running a project 'Offline', then pressing the 'Enter' key on a keyboard will start all 'Green Flag' scripts. If you *shift-click* the 'Green Flag', then 'Turbo Mode' will be activated; whilst a 'Control-click' will mute the project so that a project cannot produce any sound. The opposite of the 'Green Flag' icon is the 'Stop Sign' icon and if you click it, then a project will be halted and stop all scripts. If you add the 'Cap' block 'Stop ()' to a script and set it to 'all' then this has the same effect as the 'Stop Sign' icon, halting a project and fully-stopping all scripts.



Apart from the basic dice-thrower exercise on page 67, all of the programming exercises demonstrated so far have involved mBot and thus the need for *robotics programming blocks*. It has been mentioned in every exercise too that you must therefore be on the 'Devices' tab to access these specific blocks. The 'Sprites' tab is very much for manipulating graphics on the mBlock 5 stage (its called 'Scratch Stage Programming') and as we have demonstrated, the stage can be used to display variables data and feedback data from mBot in monitor windows - see Chapter 14 (page 86) for more on these; but on the face of it, not very much more is easily possible - so how do you achieve anything else?



“ Eureka! ”

When I found out about ‘Broadcast messages’ and how they are meant to work in mBlock 5 it was indeed a revelatory experience; and in programming terms a very ‘Damascene’ moment!

Once you realise that mBlock 5 has in reality two different Scratch programming sections to use (with mostly different blocks in each of them) it is easy to see where you are going to create ‘Device’ scripts that control mBot and ‘Sprite’ programming scripts that can be used to animate realistic graphics.

The clever bit is to understand how ‘Broadcast messages’ can be used to transfer data-on-demand between the two to enable you to display sensor feedback from mBot visually on mBlock’s ‘Stage’ (in any way that you want) and also enable you to click meaningful sprites on the stage to send control commands back to mBot.

Makeblock seem to want you, the user, to find out about programming their devices yourself so when a new (completely different) application like mBlock 5 comes along then to where do you turn? There are no books available on mBlock 5 yet! The ‘GitBook’ is rather brief and there is not a lot of really useful mBlock 5 content on Makeblock’s excellent Forum as far as I am aware.

However, in late July (2019) I still seem (on this guidance front) to be “ploughing a very lonely furrow” ! * ([but see the next page](#)) *****

In Scratch programming, scripts are written to programme an individual sprite and *are available to that sprite only*; and that script is only visible in the editor if that particular sprite is selected. In ‘Scratch Stage’ programming, broadcast messages are used to activate scripts attached to receiving sprites from any broadcasting script attached to any other sprite.

In mBlock 5 scripts are also used to programme devices so why not, I thought, try this method to *broadcast and receive between a device and a sprite* - and it worked - **YES, it’s good to talk!**

It is indeed quite possible (with a little advance know-how) to display sensor feedback from robotics devices such as mBot visually in mBlock 5 in any way that you want by using your own graphics to create high-quality interfaces; with such data represented on the stage in either digital, analogue or alphanumeric output formats.

To do this you need to clearly understand the methodology of using mBlock 5’s ‘Broadcast messages’ which must be used to communicate between *device tab* robotics scripts & *sprites tab* stage graphics scripts ...

(This is the tab-to-tab ‘transfer-data-on-demand-by-message’ broadcasting trick)

... and there is no other way to get a device and the stage to talk to each other

(apart from using - if you really have to, the Upload Mode Broadcast Extension).



mBot and Me

a Beginner's Guide

*** I originally wrote this chapter in 1919 - in January 2020 I just happened to revisit the generally useful 'GitBook' mBlock 5 Help Document which can be found at:

<https://www.mblock.cc/doc/en/>

This had clearly been updated since the last time that I visited it (- it was dated October 2019) and under 'Device Programming Basic Operations' / 'Interact with Sprites' I found the following section:

"In mBlock 5, the script editing area for devices is separate from that for sprites. To implement the interaction between devices and sprites, for example, enabling a sprite to tell the value measured by the ultrasonic sensor of mBot, you need to use the broadcast functions to transmit and receive messages (instructions) and values.

Supported devices, communication modes, and connection modes vary according to broadcast mode.

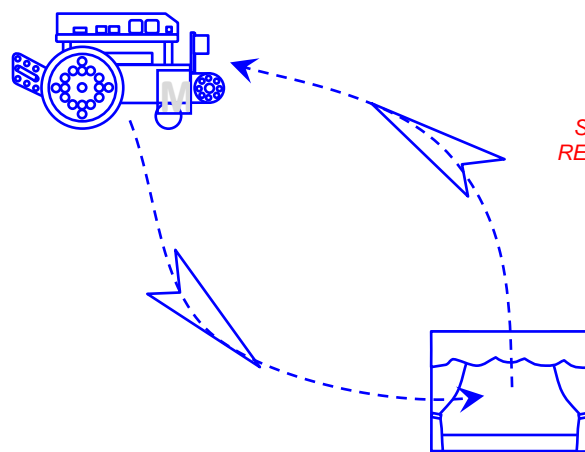
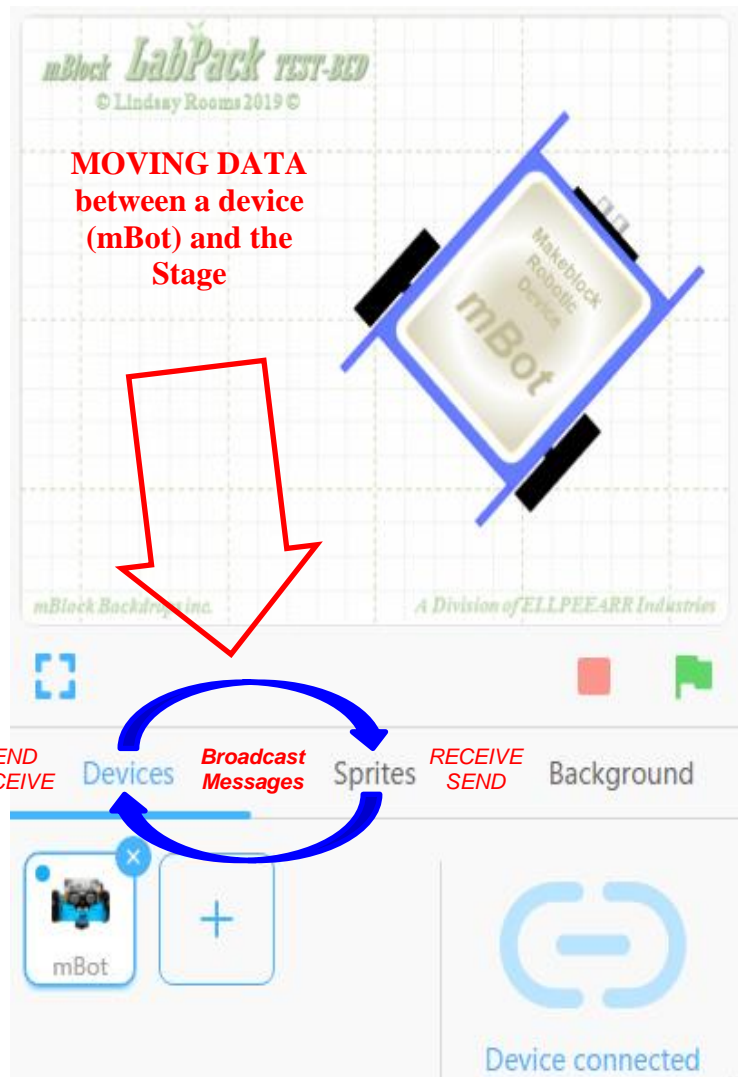
In the Live mode on mBlock 5, devices and sprites that support broadcasting can communicate with each other by broadcasting and receiving messages, and therefore the interaction between devices and sprites can be implemented".

This statement is actually quite hard to spot in the help file and although clear in what it describes, is rather brief in its description and does not make much of the significance of what I consider to be such an incredibly important concept.

You can jump straight to this section of the 'GitBook' by using this link:

<https://www.mblock.cc/doc/en/hardware-basic/interact-with-sprite.html>

On the right and below are diagrams illustrating this mBlock 5 *Devices* tab - to - *Sprites* tab 'broadcast messages' technique.





Using the tab-to-tab 'signalling' technique demonstrated in my diagrams on the previous page you can quite easily use your own graphics to create *high-quality interfaces*.

Feedback data from mBot can be represented on the 'Scratch Stage' in a variety of real-time digital, analogue or alphanumeric output formats.

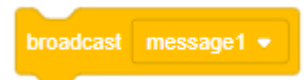
Representations of switches etc. can provide *real-time controls* for mBot too!

Using Broadcast Messages

Broadcasts are a little tricky but are nevertheless a very useful technique to learn. In Scratch, you use broadcasts to pass messages among sprites and Scratch has no limits as to where a broadcast can go.

Broadcasts are a simple but very powerful way of implementing event-driven programming and broadcasts can be used to branch a single sending script into many receiving scripts, or to close many sending scripts into a single receiving script.

The 'Broadcast' block sends messages which are used to control programme flow and any sprite (including the sprite that broadcast the message) within a project can listen for that message and respond accordingly. The stage can send and receive broadcasts too and you can have as many 'chained' broadcasts in a project as you like.



You must therefore learn to handle the development of projects running broadcasts with some care to avoid errors & subsequent mBlock 5 crashes (and consequently **you MUST progressively save and backup your projects frequently too**).

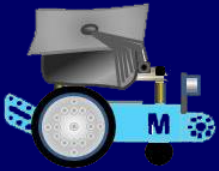
Essentially, a broadcast is a specifically-named message that is sent to activate or trigger receiving scripts after a project has started and without further user input. Broadcasts can be seen as being very similar to events called into play when certain user actions in a script (like mouse-clicks or key-presses) are performed.

Broadcasts are sent with the blocks: 'Broadcast (message_name)' or 'Broadcast (message_name) And Wait' (which waits until all activated scripts end before moving on). Broadcasts are received by the hat block 'When I Receive (message_name)'.



When using broadcast messages to communicate between a physical robotics device and computer graphics on mBlock 5's stage you need to initially prepare the way for any device data that needs to be transferred, enabling it to be stored in sensibly assigned variables using scripts written on mBlock's 'Devices' tab and then sending that data as part of a broadcast message to be received by scripts created to receive them on the 'Sprites' tab.

Got That ! ? ! - Time to move up a gear and into some very advanced work ...



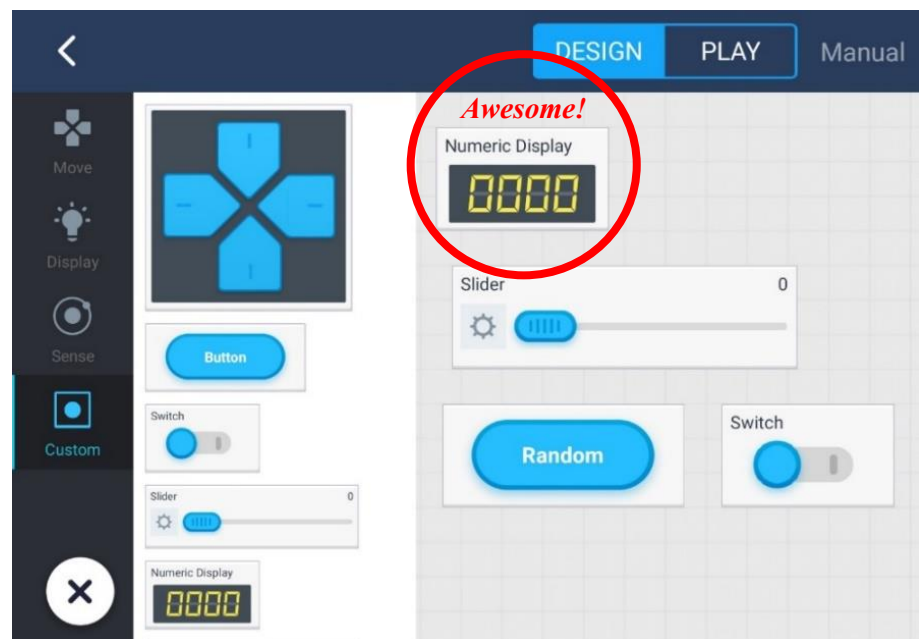
Chapter 14 - Creating Graphics Libraries for mBlock 5

I was quite enthused by the *Makeblock Android app*. when I first saw it on my granddaughter's tablet, so I have now added it my phone. Originally aimed at mBot users it now has some interaction capability with mBot Ranger.

If you click on the 'Code' option within the app. it loads '*Blockly*' (v. 0.8.5 EN) which Makeblock have developed to teach the basics of coding using drag 'n drop block programming. When loaded (according to its icon label on my phone) it seems however to be called just 'mBlock'. Blockly dates back to 2011 and whilst not quite the same visually, does resemble Scratch coding enough to move any knowledge gained within the app. across into programming in mBlock 5.

The 'Create' component of this software allows users to make their own control interface by dragging any of the components available in 'Design' mode on to the main work area from a library of components on the left of the screen.

If you switch to 'Play' mode, then these components are immediately activated, and no programming is necessary ([see a screenshot of the 'Create' screen here on the right](#)).



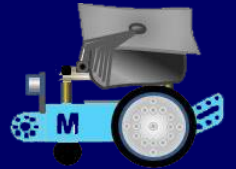
Using your own layout of switches, power-sliders and direction controllers here; together with simulated seven-segment LED displays (of four digits) which impressively show real-time numeric feedback from your device's sensors is so easy when using this component of the app.

Conversely, mBlock 5 has *NOTHING* to enable realistic robotics control / feedback interfaces to be shown on the stage display and certainly nothing like the wonderful stuff that Makeblock have developed for the Android app. mentioned above. mBlock 5 sadly has no specific device-supporting sprite libraries which can be added to the stage to produce really useful displays.

Why hasn't Makeblock developed something similar to this app. as part of mBlock 5?

It would have been really good if Makeblock had created a specific library of such stuff for robotics work rather than just the existing mBlock 5 libraries which are composed of typical Scratch 'artwork'. It begs the question, why use mBlock 5 for what can be routinely done in Scratch 3?

The backdrop and sprite libraries in mBlock 5 are typically Scratch and are NOT robotics oriented at all.



As I have mentioned before, it took me quite a long time to realise that mBlock 5 has in reality *two different Scratch programming sections* to use (with mostly different blocks in each of them). Getting to grips with this basic understanding of mBlock 5 makes it much easier to see where you are going to create '**Device**' scripts that control mBot and '**Sprite**' programming scripts that can be used to animate realistic graphics.

Creating block scripts on the sprites tab is generally referred to as '*Scratch Stage Programming*' and the blocks available for programming sprites are essentially the same as the blocks used in mBlock 5's natural parent Scratch 3. But what about 'hybrid' scripts - how can you control a robotics device *and* interact with graphical output to the Scratch '*Stage*'?

Graduating from mBlock 3 (where there weren't '*Device*' or '*Sprite*' options as separate entities) there were only one set of programming blocks to deal with and the '**Robots**' category of blocks seemed to be just tacked on to the end of the usual set of Scratch programming block categories; rather like permanently added 'extension' blocks. If you wanted to programme mBot then you just used this set of blocks with the occasional foray into the other block categories to add something useful to the '*Stage*'. Such hybrid scripts worked very well as long as you didn't want to upload them into mBot's flash memory.

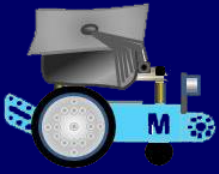
As a new mBot owner in 2018, I wanted to programme mBot as a robotics device. So other than developing robotic control scripts with a little hybrid '*Scratch Stage Programming*' added, I did not develop any of the programming skills that advanced 'Scratchers' tend to have; and so even after using mBlock 3 for a year, I was, at the start of 2019 (and switching my allegiance to mBlock 5) not an enthusiastic or skilled 'Scratcher', only understanding what I need to do to enable me to programme a robotics device such as mBot.

*It took me even longer to understand that the really clever bit of Scratch programming know-how is to understand how **Broadcast Messages** can be used to transfer data-on-demand between '**Device**' scripts and '**Sprite**' scripts. This enables you to display (in any way that you want) sensor feedback from mBot visually on mBlock's **Stage** and also enables you to click meaningful sprites on the stage to send control commands back to mBot.*

Why does Makeblock not make this vital piece of knowledge clear? As far as I understand from the excellent Makeblock support team, they are happy to resolve problems relating to their hardware or software but are not there to provide specific guidance on programming techniques.

The purpose of my book is to experiment with, understand, and explain all things mBot and I ask myself *why* (?) when it comes to graphical on-screen feedback from robotics devices, has Makeblock not given much thought to mBlock other than enabling you to develop scripts to control robotics devices such as mBot using the programming blocks (and extensions) available on the '*Devices*' tab.

Why is there nothing robotics related for the '*Sprites*' tab? *Numeric feedback from device sensors and data held in named variables (created by you) can be shown using 'monitor readouts' added to the 'Stage'*; but not much else is easily possible under the devices tab since anything further than that necessitates switching to the '*Sprites*' tab. Writing simple control scripts for mBot on the '*Devices*' tab is very straightforward (after using mBlock 3) but when it comes to the '*Sprites*' tab, you need to develop much more skill as a Scratch programmer.



mBot and Me *a Beginner's Guide*

I had to do exactly that in developing my own understanding of Scratch programming to enable me to fully utilise mBlock 5 to programme interactive controls for a robotics device such as mBot.

After many years of teaching, I am *supposedly* good at creating realistic graphics_so ...

... I decided that I needed to emulate the fine example of the Android app. described earlier and create my own comprehensive sprites library of robotics control components and several simple multipurpose backdrops on which to display them.

When creating useful stage-interface screens to work with mBot or any other robotics device you first need a suitable backdrop where you can then place switch or button sprite graphics which may need to have several 'when this sprite clicked' scripts attached to them.

It makes sense to write a simple one-line script for every sprite that you create to enable it to reposition itself (that is, if you happen to drag it by mistake!). Then switch or button sprites require a script to animate any audio-visual effects needed to suggest realism when clicked (moving either up-and-down or in-and-out etc. together with a suitable sound effect). Finally, these sprites require a script to carry-out the required action that you want to take place when the graphic is clicked (but this could be part of any audio-visual script).

Images created for the stage (whether backdrops or sprites) need to be carefully thought out and well planned. Because of the very nature of the Scratch stage's display resolution ***any original artwork that you create must be the of the highest quality possible*** to meaningfully represent and manipulate feedback from a device such as mBot.

Scribbling outline design concepts down first and creating simple sketches of your ideas is sensible. Creating an evaluation / planning algorithm (considering perhaps the specific implications of Who? What? Why? Where? and When?) for any potential display layout is almost certainly a good idea too.

But what method should you use to create the meaningful sprites and backdrops required? The obvious choice is the costume editor in mBlock 5. In it you *can* create all of the images for any project that you wish to make. It is quite powerful, versatile and generally so much better than that it was in earlier versions of the software and for many users it is now quite possible to generate reasonable looking images, either as bitmap paintings or vector based (editable / movable) drawings; but it has a major drawback ...

... If you draw anything in the costume editor then there is NO 'SAVE TO COMPUTER' option (although mBlock 5 can since its recent update export sprites as **.sprite3** files).

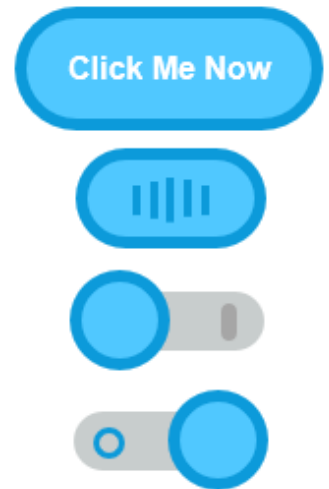
You can't download anything created in the mBlock 5 costume editor into your own filing system for future use and neither can you add them into the backdrop and sprite libraries like you can with graphics created outside the application. You *can* however **upload** your own graphics (both backdrop & sprite images) into mBlock 5. There are occasions when simple sprites can (and should) be created in the costume editor; labels for the desktop are a good and effective example; but the only sensible option is, for the most part, to **create both backdrops and sprites outside of mBlock**. Such files can then be uploaded into mBlock's backdrop and sprite libraries for repeated future use.



Graphics you create elsewhere do have to be in the right format to work, (ideally **.jpg** files for backdrops and **.png** files for sprites) but you do then have a master copy of these files in your own computer's filing system and you can more importantly upload them into mBlock's Backdrop Library or Sprites Library where they are always available for adding to any new project.

When creating both Sprites and Backdrops, **I always use the vector-graphics precision, complex colouring options, 3D extrusions and overall editing versatility of Microsoft Office**; usually using **Word**.

My first drawing test in Word was to emulate the buttons from the Android app. My drawn copies of these are shown here on the right:



You can easily capture anything you draw in Word as a bitmap (by screen-grabbing it, using Windows 'Snipping-Tool') and after a little more editing you can then upload such image files into mBlock's Sprites or Backdrops libraries for further use in new projects.

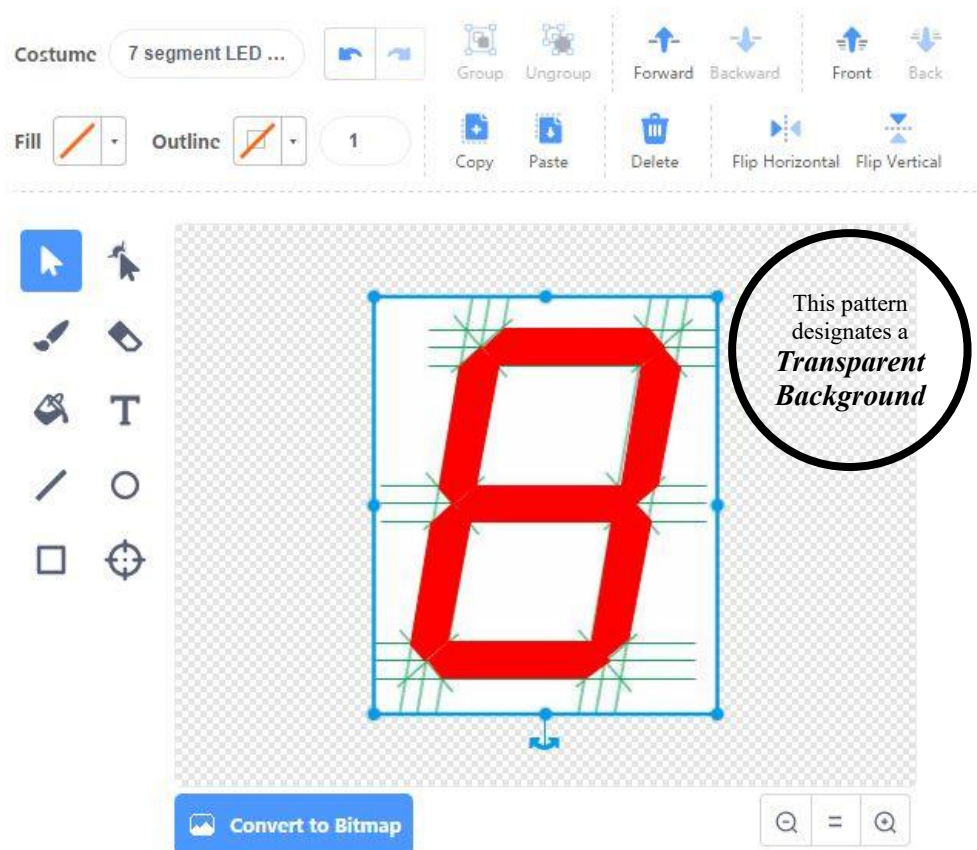
If you don't have Microsoft Office, then the free alternatives 'OpenOffice' and 'Libre Office' do work as substitutes very well.

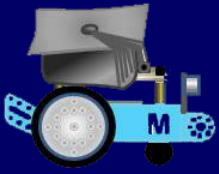
About the mBlock Graphics (Costume) Editor

The costume editor has a *transparent* background (this looks like a tiny checker-board pattern) and both vector and bitmap images created here are thus surrounded by 'nothing' when used on the stage; and all that you see on screen is what you have drawn!

On the odd occasions that you might want to draw in mBlocks costume editor, (just like in Word) **always use vectors for accuracy and editability**.

Vector drawings created in the editor can be controlled in a very similar manner to the drawing techniques used in Microsoft Office.





For the reasons outlined earlier regarding the inability to save and reuse images created in the editor this is not my preferred method; but as demonstrated, it *is* possible to create a reasonable quality 7-Segment LED graphic. A 'screen-grabbed' image of this, drawn in the Graphics Editor, is shown in the illustration at the bottom of the previous page.

In the editor, always position a sprite in relation to the default center (x, y) position; either leaving your drawing evenly spaced around the center of the sprite or (for a specific reason perhaps) with the x, y center position of the sprite set at one corner or another significant feature of the image.

If you must use the editor, then it will help considerably if you learn several basic vector drawing concepts. Clicking the pointer icon together with holding down the shift key enables you to add individually drawn vectors together and group them to become one object. You can drag the pointer too to create a 'marquee' box around a collection of shapes for grouping purposes. With an object selected, pressing the Tab key will step you forward through object sequences, whilst holding down the Shift key and pressing the Tab key will step backwards – this is in the order of object creation NOT in the current stacking order.

Holding down the Shift key whilst drawing will also constrain a line to be either horizontal, vertical or at 45°. Using Shift when drawing ellipses or rectangles will produce equal growth of the shape in x & y to produce circles and squares. Using the Alt key whilst dragging with the mouse pointer will grow shapes from the centre outwards and a combination of the Shift & Alt keys together will constrain shapes to remain both equal in x & y *and* grow (or shrink) from the centre all at the same time.

About Creating Useable Graphics in Word

This can be summarized as follows. First, you create a precision graphic using Word's powerful vector drawing tools and then capture an image of it using the Windows 'Snipping Tool' and then finally save that image as a **.jpg** file with a suitably descriptive name (**.jpg & .gif** are the only save options with the 'Snipping Tool'). At this stage the **.jpg** image that you have captured will be surrounded by the background of your Word page (usually white by default) and this colour needs to be made *transparent* for sprites to work effectively.

To do this, you need to load the **.jpg** file you have captured into a graphics editing package such as Photoshop and use the 'Magic Eraser' tool to set any unwanted white background into 'transparent'. (*If you don't have Photoshop, then free alternatives such as 'Irfanview' and the 'GIMP' will convert .jpg files to .png format and both can save the transparent attribute*).

A transparent background in Photoshop has a tiny checker-board pattern (just the same as the background shown in the mBlock graphics editor diagram on the previous page). Zoom in to check the edges of the shape very carefully and erase more background if needed.

Use the 'Clone Stamp' to edit individual pixels if further work is necessary and to touch-up any damaged areas especially around the perimeter of the required shape. You need to spend some care and time using both of these tools to prepare a sprite since any background un-erased will show when the sprite is used in mBlock 5. To retain the transparency around the sprite when it is uploaded into mBlock 5, you *must* save the file as a **portable network graphics (.png)** format file - this is the only graphics file that mBlock will accept.



About the mBlock 5 v 5.1.0 stage interface

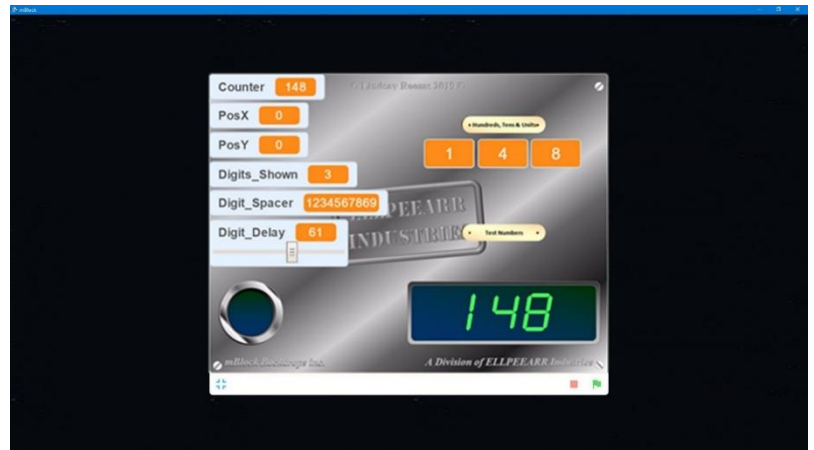
An updated version of mBlock 5 was released in early July 2019 with a modified user interface which generally has been improved. On the 'Edit' page, the 'Scripts' area is larger and there are now just two choices to adjust its size (and therefore the 'Stage' size too). A sad loss from the 'Menu' bar however is the button giving instant access to 'My Projects'. To choose a project you now have to (marginally more slowly) select 'Open' from the 'File' menu to access the 'My Projects' screen; but this is actually OK once you get used to it.

mBlock's stage backdrop size still retains the somewhat *old* standard-screen proportion of 4:3 (and a paltry resolution of 480 x 360 pixels).

How can computer graphics in the 21st century be of such low resolution?

'Presentation Mode' only doubles the display size of the stage, and bitmap graphics look somewhat pixelated in this mode, but it does not ever go to full screen height - *why?* - let alone adjust to the proportions of a modern widescreen monitor and the same is true for both mBlock 5 & Scratch 3.

Makeblock say that the interface now has brighter colours and yes, I would agree.

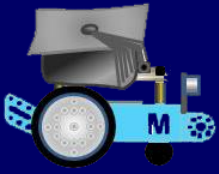


Both the 'Edit' page and full-screen 'Presentation-Mode' seem to be sharper brighter & clearer. Sadly, the area surrounding the 'Presentation-Mode' screen is no longer dimmed down (to nearly black) as it was in version 5.0.1. Note the full-screen 'screen-grab' above showing the limited size and proportion of the 'Presentation Mode' window on a modern widescreen monitor).

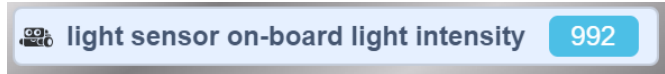
Making the surround totally black would have been a much better option. In the new release, the surrounding screen is only slightly dimmed which does have the advantage of seeing a looping script 'pulse' etc. but on the whole it is not really a lot of use and as you can see, does distract from the main event (the bit that you do want to see) the presentation screen in the centre. A second full-screen 'screen-grab' showing the new slightly dimmed 'Presentation Mode' screen on v 5.1.0 is shown here on the left.



A suitably designed backdrop is all that is needed if you wish to display data in the default variable or sensor monitors that mBlock provides. As can be seen in the top-left side of the presentation screen image here and above, data can be shown on the stage using default (standard size, labelled) 'monitor readouts'.

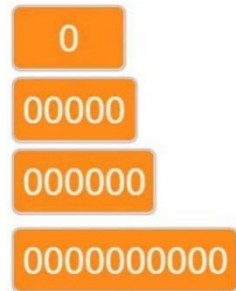


Data monitors such as these display values held in named variables that you have created. In mBlock 5 there are now monitors that can display device sensing feedback in a similar fashion with their screen output displayed in blue monitor windows.



Data monitors are useful when developing and testing projects but are rather ugly to look at; and as you can see, increase in size dependant on the length of the variable (or sensor's) name.

Note too that the monitor readout windows are marginally smaller if there is no data at all stored in the variable and they do increase in length if more than four characters are stored (as demonstrated on the diagram on the right). A double mouse-click on monitors shown on the stage toggles in turn through each of three different monitor readout variants. These variable monitor readout windows are always orange in colour with a grey rim hinting at a recess.



As can also be seen on the presentation screen images on the previous page, I used three 'large' variable monitor readouts - the ones with no name label (showing the numbers 1 and 4 and 8).

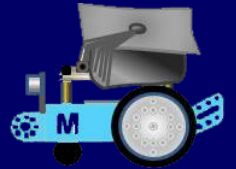
These large, label free, monitor windows are visually rather more useful, BUT they only remain in the large setting state in the stage display as long as the mBlock 5 application is open. If you close and exit mBlock 5, then saved projects lose their 'large' setting variable displays when mBlock is restarted and a project reopened, reverting back to the default, standard size monitor (with their labels showing) although they do seem to remember where on the screen they were positioned.

This seems to be a minor programming error from Makeblock here since large monitor readout settings do save and reload correctly in *Scratch 3* projects. This error was there in v5.0.1 and still exists in the new update (v 5.1.0). Projects still do not remember which of the three monitor windows were last active when they are reloaded in a new software session. They do remember where they were positioned but they always default back to the named (labelled) monitor type. I quite like using them but sadly (unless Makeblock correct this soon) this makes the use of large monitor readouts unusable in a project.

The third choice of variable monitor readouts is the 'slider' (an example of one of these is also shown in the image on the previous page). This variant does have some uses, but in both mBlock 5 and Scratch 3 they are now limited to a range of 0 to 100 (unlike in its predecessor mBlock 3) where you had the extremely useful option of being able to set your own min. & max. values for the slider adjustment of variables.

N.B. All three variants of the variable monitor readouts mentioned above are hard to position *accurately* on the stage backdrop because they can only be moved by dragging them with the mouse (there is no x or y positioning for monitor windows like you can set for sprites). **Tip:** Create a temporary sprite with a vector drawn horizontal (or vertical) line and use it on the stage to help align other sprites or monitor readouts; and then delete (or just hide) the temporary sprite on completion. **For info.:** the 'large' variable monitor readouts (the ones with no name label) measure 62 x 36 pixels in size.

You may have noticed in the same screen-grabbed images shown on the previous page that **it is possible with the right know-how to create a seven-segment LED display monitor of your own design using carefully drawn sprite costumes and some simple programming to match sprite costume graphics to numbers held in data variables.**



Seven-segment LED displays are commonly used in digital clocks & calculators etc.

Without the need to show any of mBlocks graphically limited monitor readouts on the stage at all, sprite graphics that you create yourself allow you much more flexibility (in both size and position) when a project requires you to echo the contents of stored variable or sensor feedback data. Home-grown high-quality graphics used to simulate alpha-numeric data input or output controls does give to any project shown in 'Presentation Mode' on the 'Stage' a very professional looking interface.

Can you save your graphics elsewhere?

If you want to save & move your carefully crafted projects and libraries to another computer or other storage device, you can - in a roundabout sort of way because they are all stored in the following path:

C:\Users\your username here\mblock

This folder contains several sub-folders which are updated every time you do any work in mBlock 5. Two of these folders are named 'sprites' and 'projects'.

The 'sprites' folder has four .json database files (backdrops, costumes, sounds & sprites) and show as thumbnail images of all of the graphics that you have created.

Files are shown in these library folders as thumbnails, but with unique names written in hexadecimal code. The .json files seem to be an index of mBlock's libraries of uploaded files.

You could, if you want to, just copy just the two 'sprites' and 'projects' folders to use elsewhere (on another computer or as a backup archive) but you could also store the entire mBlock folder instead, as a backup, if you think that you might need it.

N.B. Files loaded into a computer from the Cloud do not add their libraries to that computer.

New to mBlock 5 in the v 5.1.0 update (*but already available in the launch version of Scratch 3*) and a massive improvement is the much needed **'Export' sprites facility** enabling reuse of the block code and multiple costumes for that sprite when uploaded into other projects. Any attached (or floating) comment call-out notes remain attached to the saved sprite too.

If you right-click on any sprite (or the device icon in the 'Devices' tab) then 'Export' is an available choice. This enables you to save a single sprite or device as a **.sprite3** file. These can then be added into another project by using the 'Upload' button in the sprites library.

N.B. A sprite containing block scripts uploaded this way is NOT however added into your 'My Sprites' library. If the uploaded **.sprite3** file was a 'Devices' tab device icon, then it opens as a new device icon on the devices tab (remembering which device type - mBot or Codey etc.) to which it was attached. ***This is a very quick way to start a new project with some useful device stuff and sprites already in place!***

N.B. It's a very good habit to always save individual sprites which you consider to be reusable / modifiable.



Creating a Backdrop Screen Library

Using mBlocks default white stage background is boring - so, what does constitute a good graphic backdrop as an environment for robotics control? ... *what do you use as a backdrop for these things?*

Whatever you want, I guess?

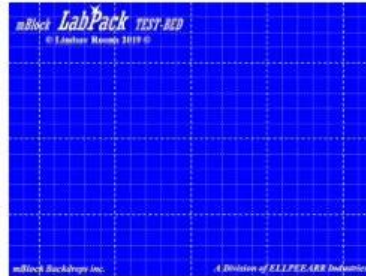
I created ten specific but fairly neutral backdrops for robotics control by drawing them in Word and then 'screen-grabbing' the resultant images.

On the face of it, the only method of presenting device feedback on top of mBlock stage backdrops is by using mBlock 5's useful but rather limited output monitor windows that can display variable and sensor feedback ([see much more about these on page 86](#)).

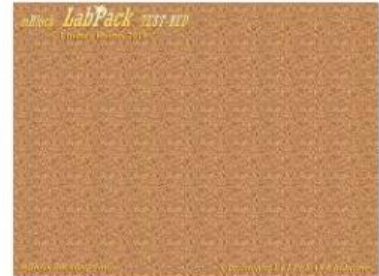
This is fine for many projects and you can choose whether to hide or show these variable monitors on the stage by clicking the tick box in the variable panel next to their names or by right-clicking a monitor and selecting *hide*.

I create my Backdrops on a Word page which had been turned into landscape mode and the margins set to 3mm. On that page I draw a rectangle 240mm x 180mm to representing the exact proportions of 480 x 360 which is the pixel display size of the Scratch stage (an equivalent of 1/2 mm to one pixel).

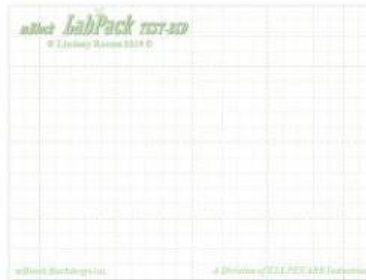
I created the ten different and easily switchable (when uploaded into the backdrops library) rectangles (shown above right) to use as 'test-bed' screens for robotics work in mBlock 5.



Blueprint.jpg



Cork.jpg



Graph Paper.jpg



Metal.jpg



Monitor Window.jpg



Papyrus.jpg



White LPR.jpg



Windows Blue.jpg



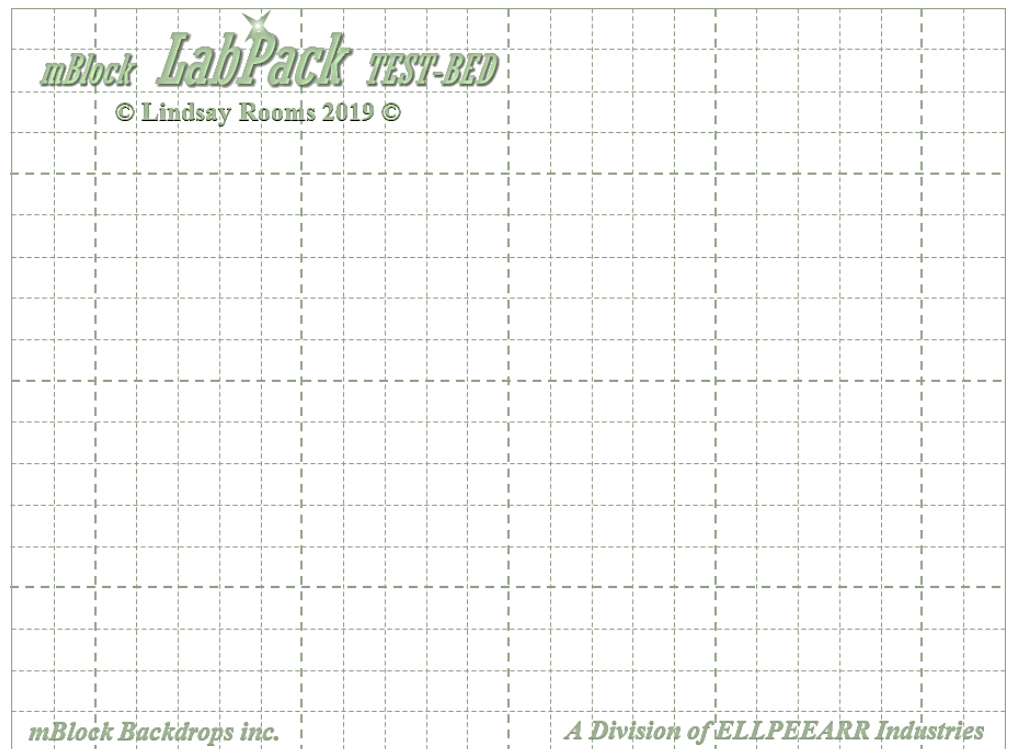
Windows LPR.jpg



Wood.jpg



The backdrop that I liked most (and decided should be my default 'set-up' screen when mBlock 5 opened) was made to look like white graph paper with light green dotted lines dividing the screen into 20 pixel squares and slightly thicker lines every 100 pixels (shown here on the right):



This looked good and indeed rather useful for helping to position sprites accurately when planning the layout of interfaces.

I decided that adding some default text top and bottom using WordArt gave it an air of authenticity.

To create files suitable for uploading, I captured them straight from my Word page using the Windows snipping tool; and to make sure that they were exactly the right dimensions I loaded each 'Snip' into Photoshop and used the cropping-tool set to 480 x 360 pixels to crop them to an exact size and resolution before resaving them as **.jpg** files.

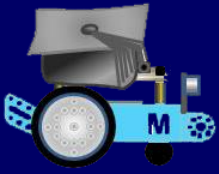
I next created a similar backdrop in blue with white lines which I called 'Blueprint' and then eight more, several of these making use of some of the different texture fills available in Word. I chose some of the textures (like 'Cork' & 'Papyrus') because they were sympathetic to the orange shade of mBlock 5's variable monitors. But, as mentioned earlier, I am no longer using these monitors since projects currently lose their 'large' setting variable displays when mBlock is restarted and a project reopened.

Creating a Robotics Sprite Library

Don't be daunted by this. It's much easier and much quicker to complete than you would think, so just be positive - but *DO* aim for a high-quality graphic *EVERY* time!

First, you need to make a master set (1 to 9 & 0) of LED digits - the method for drawing these is described on the next few pages. Once you have these drawn and converted into **.png** files, they can be inserted as costumes into a "digit" sprite and the sprite duplicated as many times as you need for different monitors. Your original sprite can also be exported and reused over-and-over again in other projects too.

Monitor windows of various proportions are also needed and then you can develop switches, sliders indicator lamps, direction controls etc. etc. as you need them.



Creating 7-segment LED display numeric digits

As mentioned earlier, is quite possible to bypass variable & sensor data being displayed in any of the three monitor readouts variants described. It's much better to display such data using LED type graphics of your own creation; any size, any style, any colour, positioned accurately anywhere on the stage.

To create your own data monitors, you need to create individual sprites for digital 'Units' (or 'Tens', or 'Hundreds' etc.) with ten individual number costumes added to each sprite. Creating a set of ten LED digit costume drawings initially seems to be a rather tedious to create, but the work is worth it when they can be used over-and-over again. Also, and rather complex, (but nevertheless achievable) is the programming required to match a sprite (and its costumes) to a digit stored in any variable that you wish to display on the stage.

You can just type individual digits into a sprite in the costume editor (one costume each for digit 1 to 9 and 0). This is a very simple but effective method. You can also capture .jpg images of alphanumeric characters in any font, style and colour too and these would also be quite suitable for many projects.

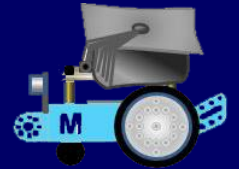
However, traditional seven-segment LED display characters look particularly good when showing device feedback in mBlock 5, especially if they are placed in front of graphically drawn feedback monitor windows on an equally carefully thought-out stage backdrop.

Real electronic LED displays are available in a variety of LED colours; most notably Green, Red, Yellow, Purple & Cyan. The Scratch programming 'change colour' block can be used to change the colour of any sprite and the 'brightness' of graphics can also be changed using the 'change brightness' block to simulate fading or flashing displays. Seven Segment Displays are still very common but are now often superseded by 8x8 LED matrix displays or LCD displays which produce finer display characters. LEDs are often in a proportion of 5:3 and typically slope by approx. 10° (*italicised*) but they can be found as vertical digits too.

As shown in the diagram on page 83, it is quite possible to create these in the mBlock graphics editor; this diagram shows that if you go to 'Costumes' and create (using vectors) a basic grid in a new sprite and draw just one horizontal & one vertical segment of the seven segments required. You can then just copy and paste each of these enough times to make up the seven segments required for the numeric digit "8". This is the first costume within a new sprite, so you need to simply name it '1' (even though it looks like an '8' and then duplicate it another nine times in all to create a total of ten costumes for the sprite & they should have been automatically named 1 to 10. Name the sprite itself 'Digits'. Next, edit each costume graphic in turn to delete the segments not required to create each number so that each costume matches its name (& rename the last costume "0").

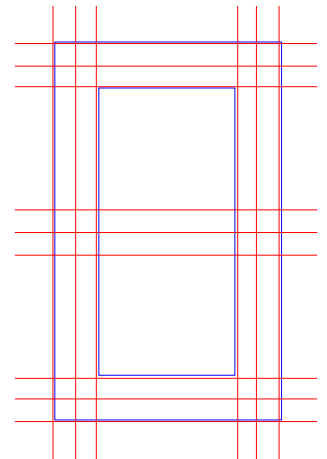
As mentioned earlier too, using the Graphics Editor is rather pointless since *any graphics created here cannot be reused easily*. It is therefore far easier to create much more precise LED digits using the drawing tools in Word which can then be converted to the correct format and uploaded into your sprite libraries.

Using Word to do this is much quicker than you would think since numeric digit "8" contains all seven segments of a LED display and all the other number costumes can be made by duplicating this one and deleting the segments not required for each number. (See the drawn diagrams on the next page).



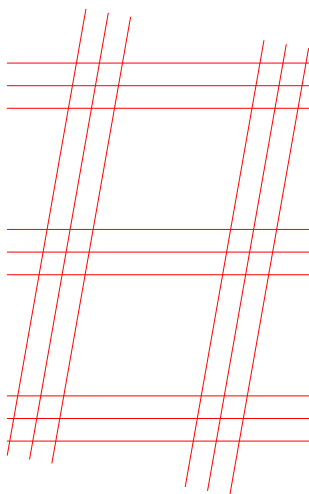
First you need to first create a precision construction grid; eventually sacrificial, but nevertheless important in developing an accurate technical drawing. Draw two (thinnest line possible) rectangles one 50mm x 30mm and the other 38mm x 18mm and middle them horizontally & vertically. Group them together & make the line colour blue; these signify the outer and inner dimensions for your drawn characters enabling 6mm thick individual segments to be drawn to create the component segments of a LED graphic.

Draw a vertical red line, with its Height set to 60mm and then draw a horizontal red line with its Width set to 35mm. Duplicate these lines five more times to make six of each in all and position them on top of the two rectangles as shown in the diagram on the right.



You need to set each group of three lines (both horizontal and vertical) to be 3mm apart in each direction. Use 'Middle Horizontally' and 'Space Horizontally' to help you do this to the vertical lines and then use 'Middle Vertically' and 'Space Vertically' for the horizontal lines. Finally group all of the lines together and you can then, if you want to, delete the blue rectangles.

You should by now have an accurately drawn red-line grid that looks like the one shown above right.



To create the slightly more sophisticated *italicised* look (and this is the style that I personally think looks best) you need to start by duplicating the grid you made earlier; ungroup this duplicate grid and rotate all of the **vertical** lines by 10°. Regroup the new grid and it should look like the diagram shown here on the left. You now have an upright grid and an italicised grid for creating seven-segment LED graphics. Save your file.

Return to your original upright grid so that you can create the required segments. To obtain greater accuracy increase the zoom factor of the Word page so that your drawn grid fills most of your screen.

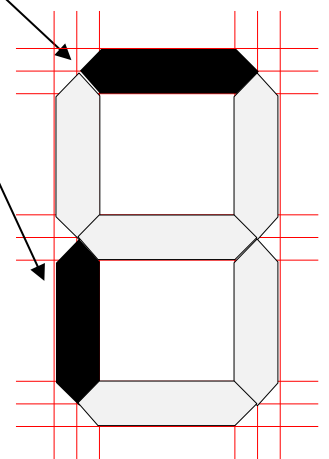
Select the Freeform Shape tool and click it in the centre intersection of the topmost left-hand corner of your grid and draw a segment like the horizontal one shown in the diagram on the right.

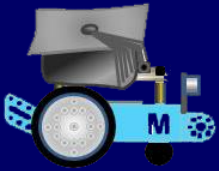
Next draw a vertical segment like the one shown below it.

Duplicate these as necessary to complete all seven segments of an upright LED "8" digit.

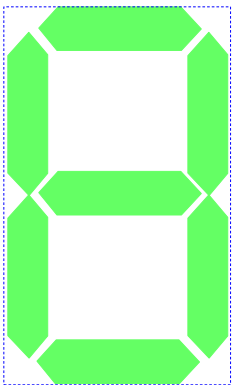
This is the only digit you need to create to make a complete set of upright number digits since it contains *ALL SEVEN* segments of the LED display.

All the other digits can be made by duplicating this one and deleting the segments not required for each number.





mBot and Me *a Beginner's Guide*



To look like a real LED display, the segments now need to be separated slightly. Leave the centre segment where it is and use the cursor keys to nudge the remaining segments as follows:

Top & Bottom segments (three nudges outwards)
Side segments (four nudges outwards).

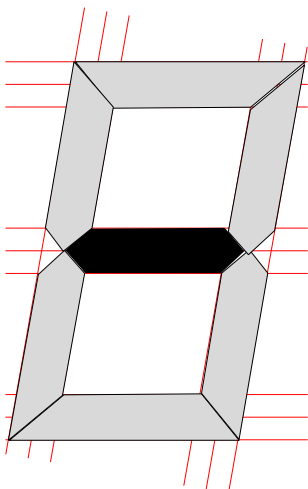
This may sound quite a lot of nudges, but as you can see in the set of digits (on a black background) shown below-right, the smaller the LED digit, the less noticeable are the gaps between the segments.

It makes sense to shrink segment-separated (*stretched*) digits back to their originally designated 50mm x 30mm size. Use another temporary 50 x 30 rectangle to do this (see the diagram above).

Group all seven segments together and duplicate them nine times in all to make the complete set, deleting the relevant segments from each digit as required. Align all 10 of these digits into a horizontal line and space them evenly.



You don't have to do this bit, but if you add a temporary construction line at the top and another one at the bottom of the row of digits you will see that the vertical segments in the digits 1, 4, 7 and 9 do not reach either the base line or the top line (in reality, this is what would happen with the segments of a real LED display). I am inclined to follow this principle, leaving the height of these segments as they are - but you might wish to make them all the same height; so, just stretch each segment of these four digits vertically to touch the construction lines (see above) and when this is done, delete the two lines.



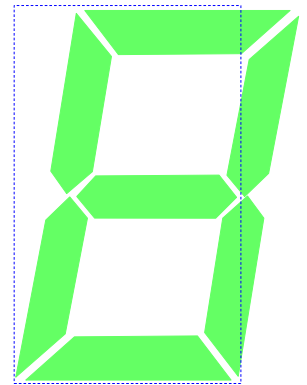
To create my preferred (*italicised*) LED digits, you need to repeat a process similar to the one described for creating upright digits, but this time using the second (*leaning 10°*) red grid that you made.

The middle segment of this new digit can be copied from your vertical LED drawing. Once again, use the Freeform Shape tool and create the remaining six new segments as shown in the diagram on the left.

Remember to zoom-in as much as you can to draw these remaining segments. Accuracy when drawing segments on top of the grid is vital.



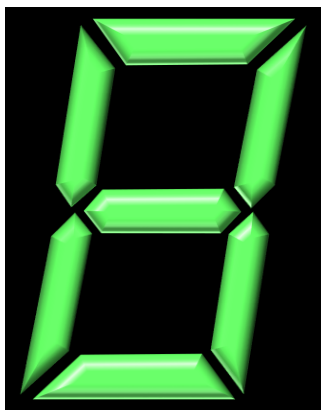
Modifying your drawn shapes using Edit Points is an important skill to master too and do not be afraid of deleting a shape and completely redrawing it again if any segment does not look right. Finally, remember to nudge the segments apart and then shrink the group of segment-separated (*stretched*) digits back to their originally designated size.



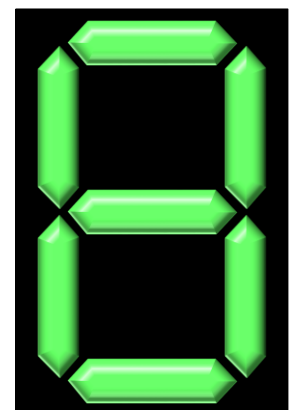
The outline of all of the drawn segments of my two basic green LEDs are set to *NO Outline* and the flat-fill colour that I have used is demonstrated here on the right. When creating these drawings in Word, the *HSB* colour fill settings are: **Hue: 85, Saturation: 255, Brightness: 180** and their equivalent *RGB* settings are: **Red: 100, Green: 255, Blue: 100**.

However, when editing in Photoshop, colour fills are vastly different to the settings to those in Word (described above) and different again in mBlock's own graphics editor. Photoshop's equivalent settings to the shade of green I used in Word are: *HSB* **Hue: 120, Saturation: 80, Brightness: 100** and *RGB* **Red: 50, Green: 255, Blue: 50**.

To create digits with transparent see-through backgrounds you *do* need to edit them in a graphics editor such as Photoshop and save them as a **portable network graphic (.png)** file. This file type is commonly used to store graphics for web images (where they always need to have a transparency attribute set).



You can see that in the image on the left that the outer corners of the italicised LED digit that I created look quite sharp and ideally should be radiused slightly; but this would be hard to achieve using the vector drawing methods described earlier. *BUT* - it is possible to do this when converting the drawing into a .jpg painting file which you can edit in Photoshop.



I found that the best way of creating a full set of ten numeric digits (0 to 9) was *not* to create them by duplication and editing in Word but to create just ONE seven-segment LED number "8" digit (one

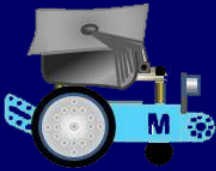
italicised and also if required, one upright - see the two images above).

I found that adding a black background behind (and just surrounding) my two "8" digit drawings worked well enabling me to check my visualisation modifications clearly. To create the two images above, I set a 'No Outline' attribute for both and applied several 3D transformations which gave them a much more realistic "illuminated LED" look. These transformations were:

Bevel 6 x 6, Depth 6, Contour Dark Green, Material Plastic and Lighting set to 3-Point.

These seemed rather more realistic than the flat-green early versions (although they look good too).

Using the Windows Snipping Tool, I captured each of these "8" digit drawings on their black backgrounds in turn and saved them as .jpg files with a suitably descriptive name (remember, .jpg or .gif are the only options).



mBot and Me *a Beginner's Guide*

I named each of these files *Digit (italic) 8* and *Digit (upright) 8*. I then created two file folders called *LED Sprites (upright)* and *LED Sprites (italic)* and moved each of my two newly created .jpg files into their matching named folders.

In each folder I copied the original *Digit upright / italic 8* file and pasted it back into the folder until there were nine copies of the original "8" digit file in each folder. I then altered the name of each so that the number at the end of each filename was from 1 to 0 (even though at this stage they all had the same "8" graphic) and each one still with a surrounding background colour of black (which eventually needs to be made transparent). To create this transparency, I loaded each .jpg file in turn into Photoshop and used the 'Clone Stamp' set to mimic the black background and then painted-out the unwanted segments for each digit.

When each number looked as it should (matching it's file name), I switched to the 'Magic Eraser' tool and used this to set the unwanted black background as transparent (the tiny checker-board pattern). I used the 'Zoom' tool to check the edges of the shape very carefully and erase more background if needed. Mostly this worked perfectly much of the time. I also used the 'Clone Stamp' again to touch up any damaged areas especially around the perimeter of the required shape.

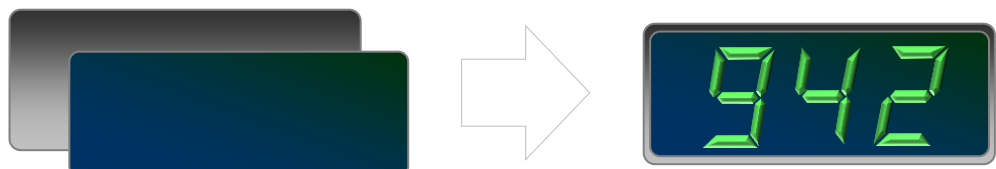
The Magic Eraser with its tolerance set to 40 worked well and I had very little further editing to do but you do need to spend some care and time using both of these tools to prepare a sprite since any background un-erased will show (annoyingly) when the sprite is used in mBlock 5.

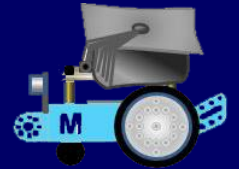
After I had edited and saved each of my numbered digit images files in turn, I opened each folder to check the thumbnails to see that I did indeed have a complete set of ten digit images (0 to 9). Finally, I reloaded each .jpg file in turn back into Photoshop and saved each one as a .png (portable network graphics) format file which retains the transparency attribute set of the sprite; and when each folder had ten individually named .png files (each showing just a unique numbered digit image with no black background) I no longer needed my original set of .jpg files, so I deleted them. You may not be brave enough to do this, so do keep them in the folder if you wish to.

Next, I turned my attention to creating a sprite image for a monitor readout window that could be added to a backdrop and positioned (in any size, anywhere). These needed to be effective as a realistic display background for my 7-segment LED images. Unlike the black windows used in the Android app. I wanted the look of a traditional LED display, a recessed dark-coloured window. To make these is very simple, just a trick of effectively using light and shade.

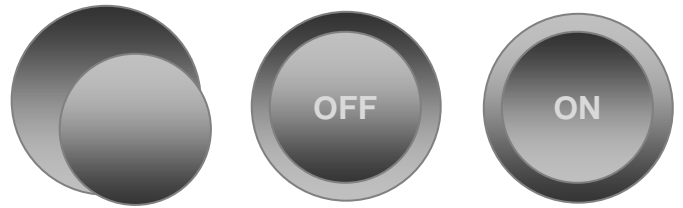
Draw a Rectangle with round corners and duplicate it to make a second matching rectangle, the second stacked on top of the first. Give the first of these a mid-range thin grey outline and set a Gradient Fill for the shape - Linear 90° - dark grey at the top to light grey at the bottom. Give the second of these the same mid-range thin grey outline and set a Gradient Fill for the shape - Linear 315° - dark-blue, bottom-left to dark-green bottom-right. Reduce the size of the front rectangle slightly (equally in x & y) and centre it horizontally and vertically on top of the first rectangle. The amount of reduction in size of the top rectangle dictates the depth of the recess (see below).

Group both rectangles together - effective isn't it!





To try something similar, draw a circle and give it a mid-range thin grey outline and set a Gradient Fill for the shape - Linear 90° - dark grey at the top to light grey at the bottom.



Duplicate it and reduce it in size. Reverse the shading, Linear 90° - light-grey at the top to dark-grey at the bottom. Centre the two circles and note how this becomes a button in a recess!

Duplicate it again and reverse the shading of each circle - now this one looks like a button that has 'popped-out'. You've made rubber buttons! Add text into the middle of each button. If you make a 'Button' sprite and upload .png images of each of these drawings as costumes into an mBlock 5 sprite, you can then use the 'When sprite clicked' programming block to change to the next (or previous) costume. You'll need to create this simple decision making script to make it work.

```

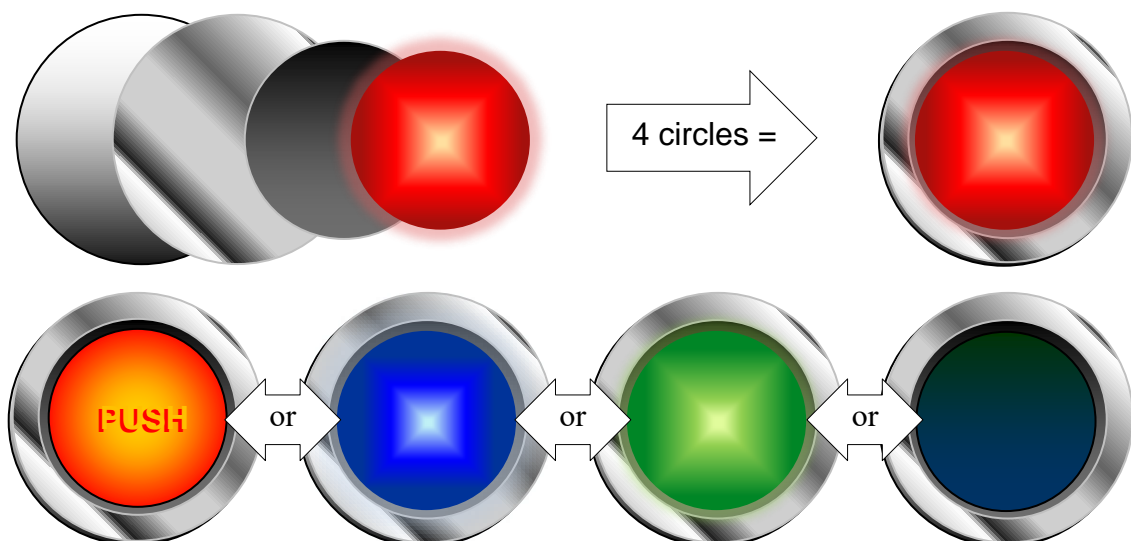
when this sprite clicked
if Switch_Pos = 1 then
switch costume to 1
set Switch_Pos to 0
else
switch costume to 0
set Switch_Pos to 1
    
```

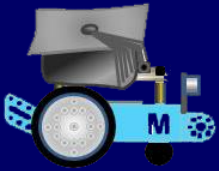
If you have tried this out then you will have created a realistic working button - especially if you have completed the illusion with a suitable sound effect ('click') and you can of course change the grey shades demonstrated above to any colour combination that you like.

Many other .png files with transparent backgrounds can be created using the methods described above. I used to use these basic skills of manipulating light and shade a lot when I was teaching many years ago to enthuse kids by getting them to make high-quality graphics which they learned to programme using VBA (in Microsoft Excel).

In an advanced version of the graphics techniques shown above I used to use an exercise outlined briefly at the top of the next page. This project was a graphical illusion, using 'visibility' to hide/show one graphic at a time; clicking one graphic hid it and showed its opposite; just a simple on/off animation trick. Kids reactions when the switch 'worked' for the first time (with a 'click' sound effect added) were always amazing; "... I've programmed something that goes in-and-out of the screen - Wow!"

Shown below is the method for creating drawings of coloured indicator lamp 'bulbs'.



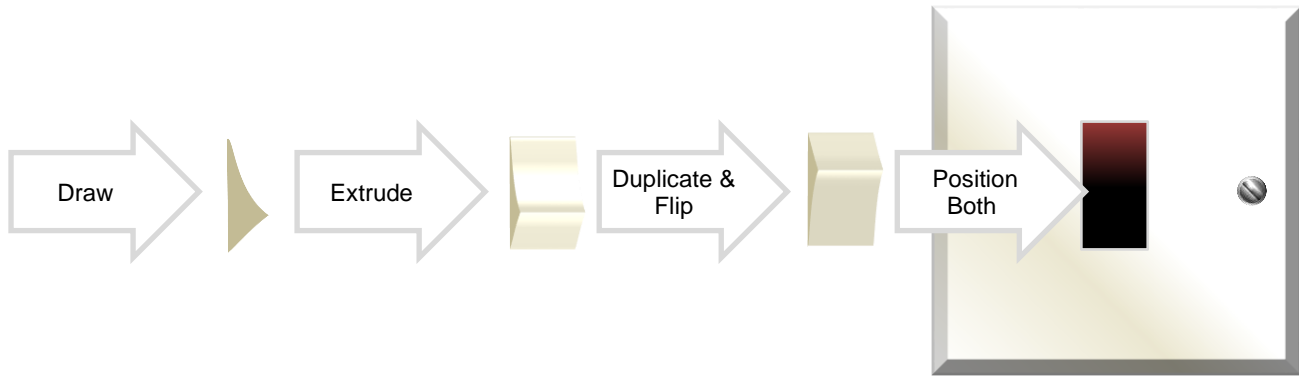


mBot and Me

a Beginner's Guide

This Excel exercise was so simple, but how effective in developing an enthusiasm for programming and for spreadsheets too (sneaky teacher!).

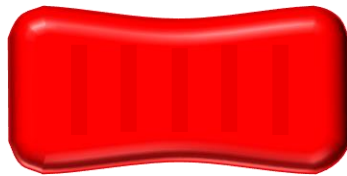
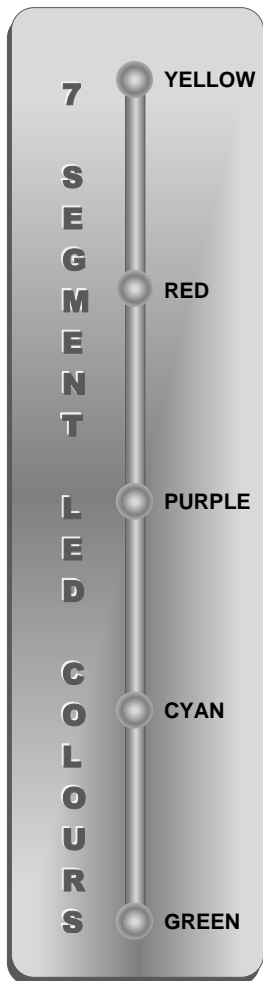
Two switch extrusions were positioned on top of each other (on a switch plate recess) and then one of the two individual VBA subroutines shown was assigned to each of the rocker-switch drawings:

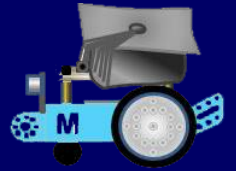


```
Sub hide_OFF_switch()  
ActiveSheet.Shapes("OFF").Visible = False  
ActiveSheet.Shapes("ON").Visible = True  
End Sub
```

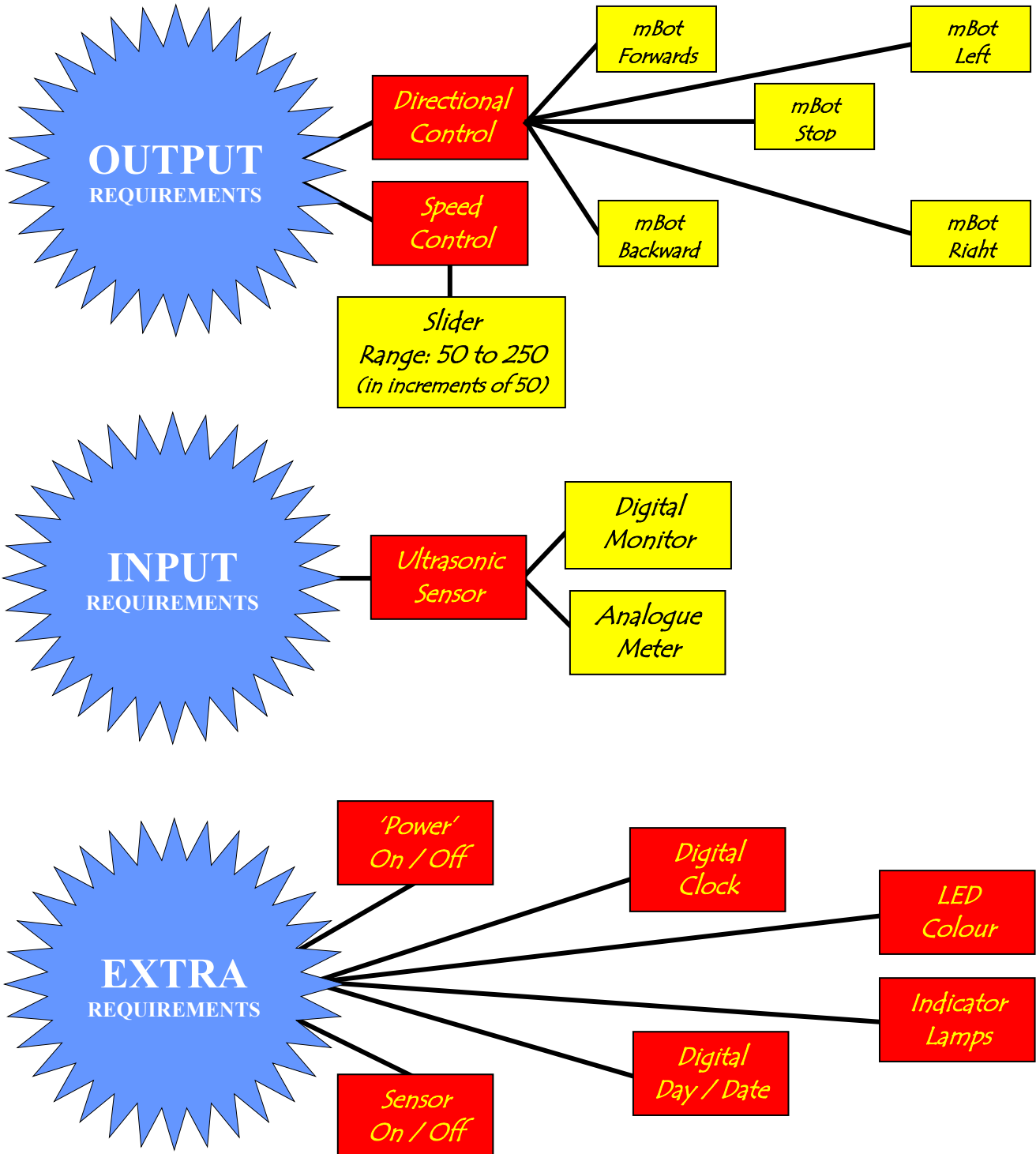
```
Sub hide_ON_switch()  
ActiveSheet.Shapes("OFF").Visible = True  
ActiveSheet.Shapes("ON").Visible = False  
End Sub
```

Some other graphics I created to use in the 'Control Interface' project described in the next chapter were:





Design Requirements
for an mBlock 5 stage
'Presentation Mode'
CONTROL INTERFACE
for an mBot robotics device





Chapter 15 - Building a Control Interface for mBot

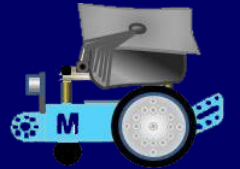
Once you realise that mBlock 5 has in reality two different Scratch programming sections to use (with mostly different blocks in each of them) it is easy to see where you are going to create **Device** scripts that control mBot and **Sprite** programming scripts that can be used to animate realistic graphics.

*The clever bit is to understand how **Broadcast Messages** can be used to transfer data-on-demand between the two to enable you to display sensor feedback from mBot visually on mBlock's **Stage** (in any way that you want) and also enable you to click meaningful sprites on the stage to send control commands back to mBot.*

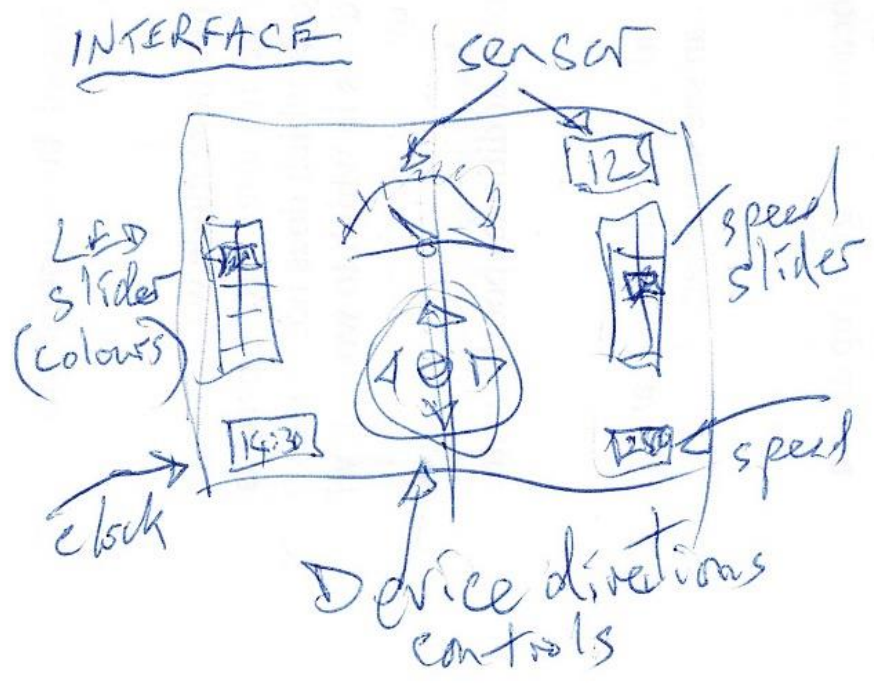
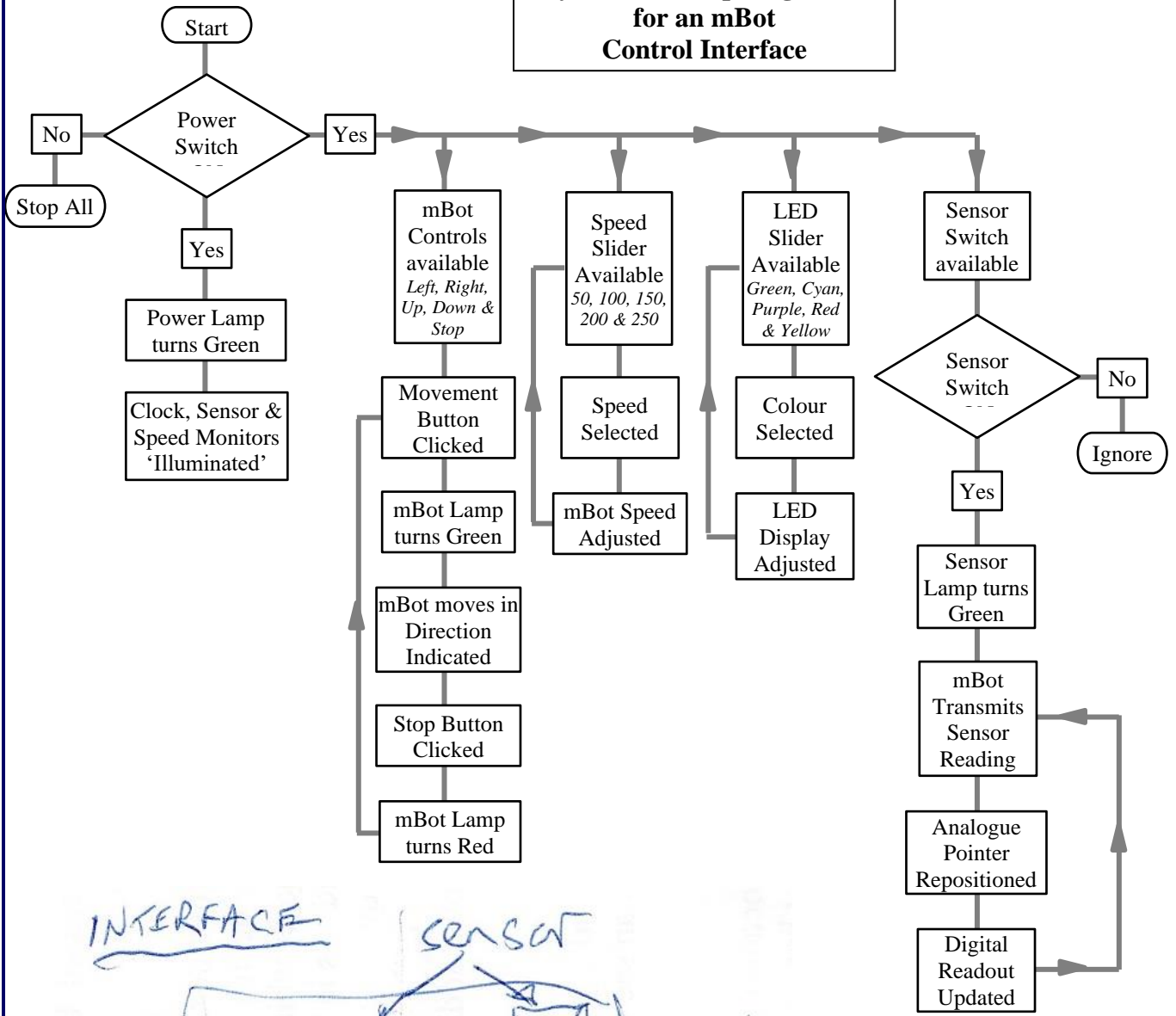
By using your own graphics to create high-quality interfaces, feedback data from mBot can be represented on the Scratch Stage in a variety of real-time digital, analogue or alphanumeric output formats; and representations of switches etc. can provide real-time controls for mBot too!

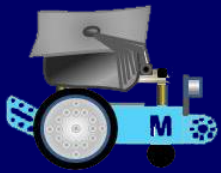


My first complete development of a fully working **Device / Stage Interface** (shown above) is explained in full detail in this chapter - however in reality each of the sub-component parts of my ideas were 'test-bedded' separately first and then added together to form the complete project which relies on using pre-prepared graphics uploaded into mBlock 5's libraries. (See how to make these in the previous chapter).



My Basic Concept Algorithm for an mBot Control Interface





Screen-shots of some of my 'test-bed' sub-component projects are shown below:

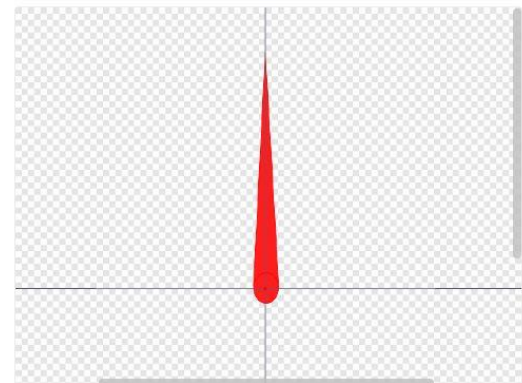


Once I had an idea of what output and input tasks were required for a fully working interface, I had to consider the visual look that I thought was needed.

I then to make sure that I developed the required stage graphics for both the display output of feedback and switch / button / slider input (see pages 90 to 96 of Chapter 14 for more on this). All of these graphics had to then be uploaded into either my mBlock 'backdrop library' or 'sprites library'.

I decided that the shiny metal plate background that I had created would be my final backdrop for the project *although initially I used my graph-paper backdrop to help me align and position the individual graphics on the stage*. Knowing from early tests that I could get the value from mBot's ultrasonic sensor stored in real time in a variable that could be used by graphics on the mBlock stage, I had in mind that I wanted both *digital* feedback and an *analogue* meter of some kind with a pointer fluctuating as the sensor values changed. This sensor displays distances of up to approx. 4M (4000mm) so I need a meter capable of displaying 4000 divisions in a semi-circle.

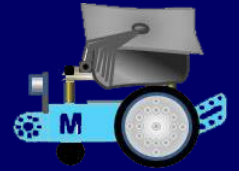
Another quick test showed me that the 'point in the direction' block taken from the 'Motion' section of the blocks menu would do this very well as long as the sprite had its axis of rotation positioned on the centre point position in the costume editor's display screen. For 180° of rotation / 4000 steps you need to set the direction of rotation to be (sensor reading x 0.45°). After this test I knew that I needed to create and add a new .png graphic of some sort to my sprite library - a monitor window to create a realistic analogue meter.



My aim was to also have a four-way direction controller for mBot and a 'stop' button too. I also planned to add two moveable sliders, one for altering mBot's speed and another to change the colour of any 7-segment LEDs used.

To go with the first of these sliders, I thought that a speed monitor would be good to show the speed set and after much deliberation decided that adding the day, date and time might be a nice feature and as a demonstration of how to add these to a project. I also planned to demonstrate a couple of different methods of producing 'clickable' buttons.

If you want to try making a similar interface, start by opening a new mBlock project file and then follow the sequence of steps beginning on the next page. **N.B. You will need to use your own (previously created) graphics libraries to achieve this.**



Step 1: - Create the required project variables. This is a marginally tedious task, but not too onerous.

Scratch variables are always displayed alphabetically, so I decided to prefix all of them with logical sub-group names (see the list on the right) so that individual variables that related to specific tasks were displayed together in the list making them easy to find when adding them to individual sprite scripts later.

Make the complete set of 24 variables as shown on the right (& they are also listed rather more clearly below):

Cal_Day, Cal_Month, Clock_Hours, Clock_Minutes, Clock_Seconds, Digit_Hundreds, Digit_Tens, Digit_Units, Digits_Shown, Digits_Size, Digits_Space, Digits_Xpos, Digits_Ypos, mBot_Speed, Plus_Factor, Sensor_Pos, Sensor_Value, Slider_Xpos, Slider1_Switch_Pos, Slider1_Ypos, Slider2_Switch_Pos, Slider2_Ypos, Switch1_Pos, Switch2_Pos.

To make these, click on the 'Variables' choice in the blocks menu and then click the 'Make a New Variable' button. Then type each of the required variable names in turn and keep the default setting 'For all sprites' for each of them.



Analogue_Pos
Change_LED
mBot_Backward
mBot_Forward
mBot_Lamp
mBot_Left
mBot_Right
Move_Monitor
Move_Slider1
Move_Slider2
Poll_Sensor
Power_Lamp
Send_Data
Sensor_Lamp
Stop_Motors
Update_Clock
Update_Digits
Update_Speed
Zero_Digits
Zero_Pointer

Uncheck all of the tick boxes next to each variable name to hide their stage monitors. This is now a good time to save your project with a suitable filename.

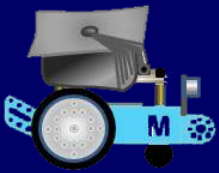
Step 2: - Next you need to create the complete list of message names that need to be broadcast between the device, mBot and sprites on the stage (& vice-versa). To do this go to the 'Events' section of the blocks menu. And use the 'hat' block 'when I receive' shown here or the 'broadcast' block (in fact, either of these will do).



Click the down arrow to access the drop-down list and click on 'new message'. Type in the complete set of (20) messages shown on the left one after another (these are also listed rather more clearly below):

Analogue_Pos, Change_LED, mBot_Backward, mBot_Forward, mBot_Lamp, mBot_Left, mBot_Right, Move_Monitor, Move_Slider1, Move_Slider2, Poll_Sensor, Power_Lamp, Send_Data, Sensor_Lamp, Stop_Motors, Update_Clock, Update_Digits, Update_Speed, Zero_Digits, Zero_Pointer

Save your project again (you do need to do this on completion of every step).

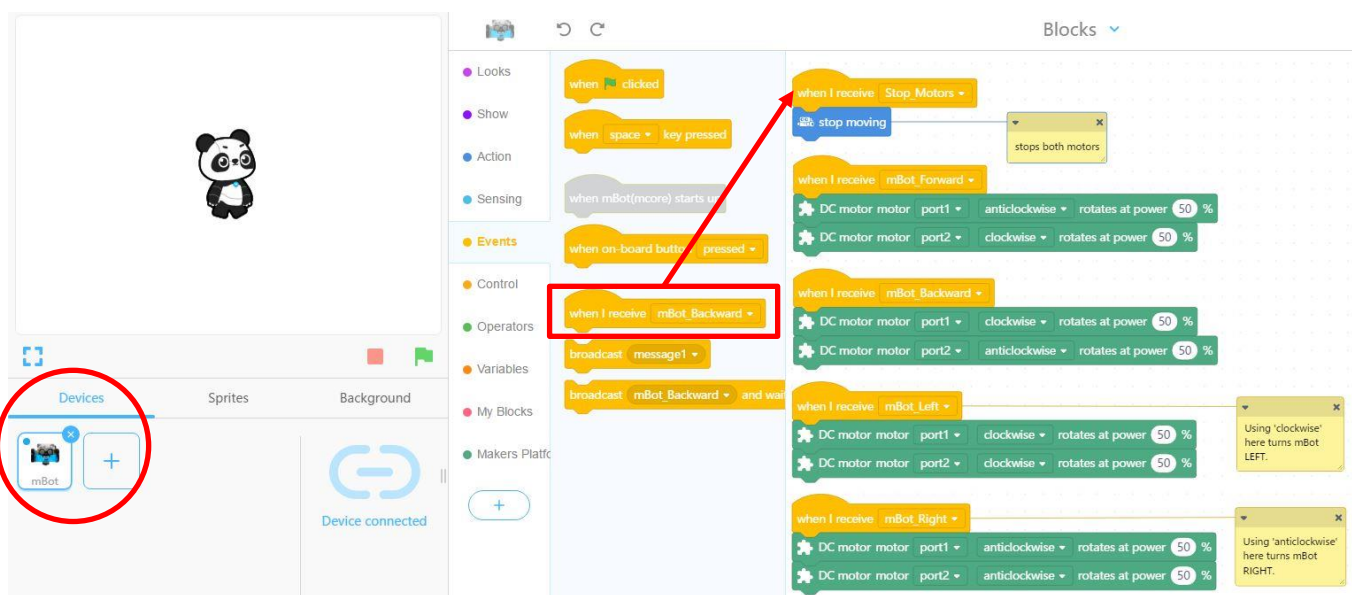


mBot and Me a Beginner's Guide

Step 3: - Now is a good time to test some of the Messages defined in Step 2 earlier by broadcasting them from the 'Sprites' tab to the 'Devices' tab to send control commands to mBot, so delete 'Codey' and choose mBot as a new device and remember to go to the 'Devices' tab and click **connect!**

If you haven't done so already, click on the 'Devices' tab. See the diagram below. Here you need to go to the 'Events' section in the 'Blocks' area and drag the 'when I receive' hat block shown above onto the 'Scripts' area on the right of the screen. You need five of these in all and you can either drag more across or duplicate four more from the original one you dragged across first.

Go to each hat block in turn and click the down arrow to access the list of messages - choose 'Stop_Motors' as the message for this block.



Next go to the remaining four hat blocks in turn and set the message for each as follows, 'mBot_Forward', next block 'mBot_Backward', next block 'mBot_Left' and finally 'mBot_Right'.

They now need control ('Action') blocks adding to each of these hat blocks as shown in the diagram above. The first ('Stop_Motors') block is straightforward. Choose the 'stop moving' block from the 'Action' section of the blocks menu and drag it across to connect with the hat block.

It is possible to use from the 'Action' section, the 'move forward at power()' block for the four remaining hat blocks; changing 'forward' to 'backward', 'left' or 'right' as appropriate. I decided however to use the rather more sophisticated action blocks (shown in the diagram above in Arduino house-colour, sea-green). These are taken from the 'Makers Platform' extension, and if you haven't added an extension pack into mBlock before, is not a bad thing to try this right now.

Leave the default power setting at (50%) for now. Eventually, but not yet, you will need to replace this percentage in all of these control blocks with the variable 'mBot_Speed'. This is needed when you have some way of changing the value of that variable (by adding and programming a slider control graphic).

Save your project once again.

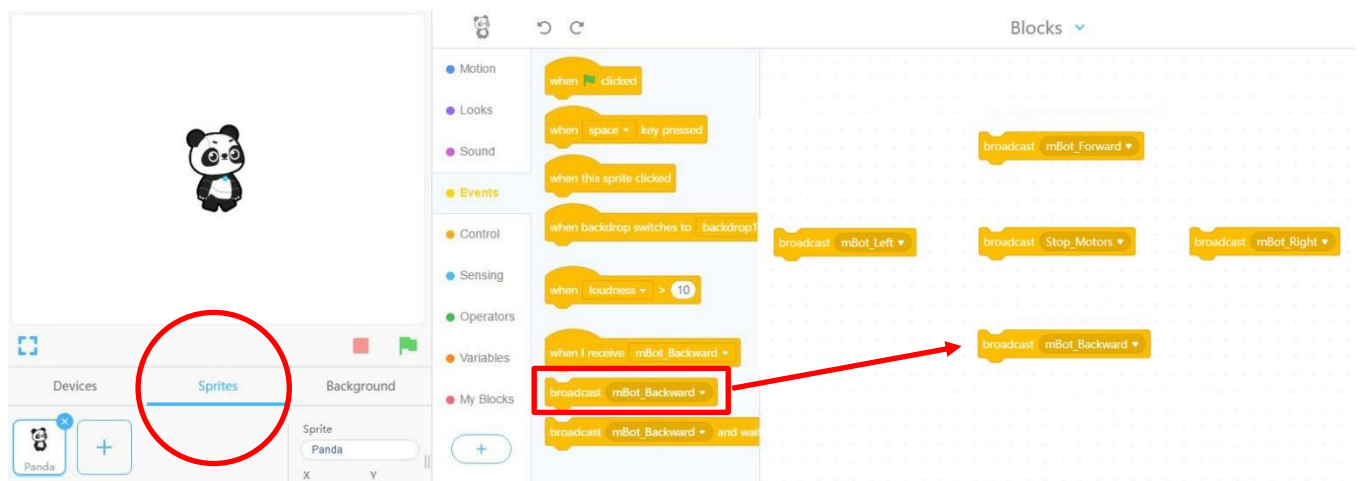


Step 4: - Click on the 'Sprites' tab. See the diagram below.

To test that mBot can indeed be controlled from the 'Stage' using sprites is actually very simple. All you need are five 'broadcast' blocks taken from the 'Events' section of the blocks menu.

Do note that blocks in the 'Events' section are identical in both the 'Sprites' tab and the 'Devices' tab whilst other block categories are not.

Give each of these 'broadcast' blocks message names matching the ones that you used for the five hat blocks on the devices tab. You will note from the diagram shown below that I arranged each of these blocks with the 'Stop_Motors' block in the centre with forward at the top, backward at the bottom and left and right positioned accordingly. NO (green flag) hat blocks are needed above these to make this work.

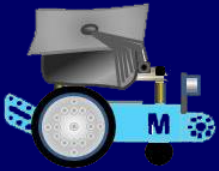


Just try clicking each broadcast block in turn to activate it. If mBot is connected, then it will move accordingly and prove that sprites scripts can control devices!

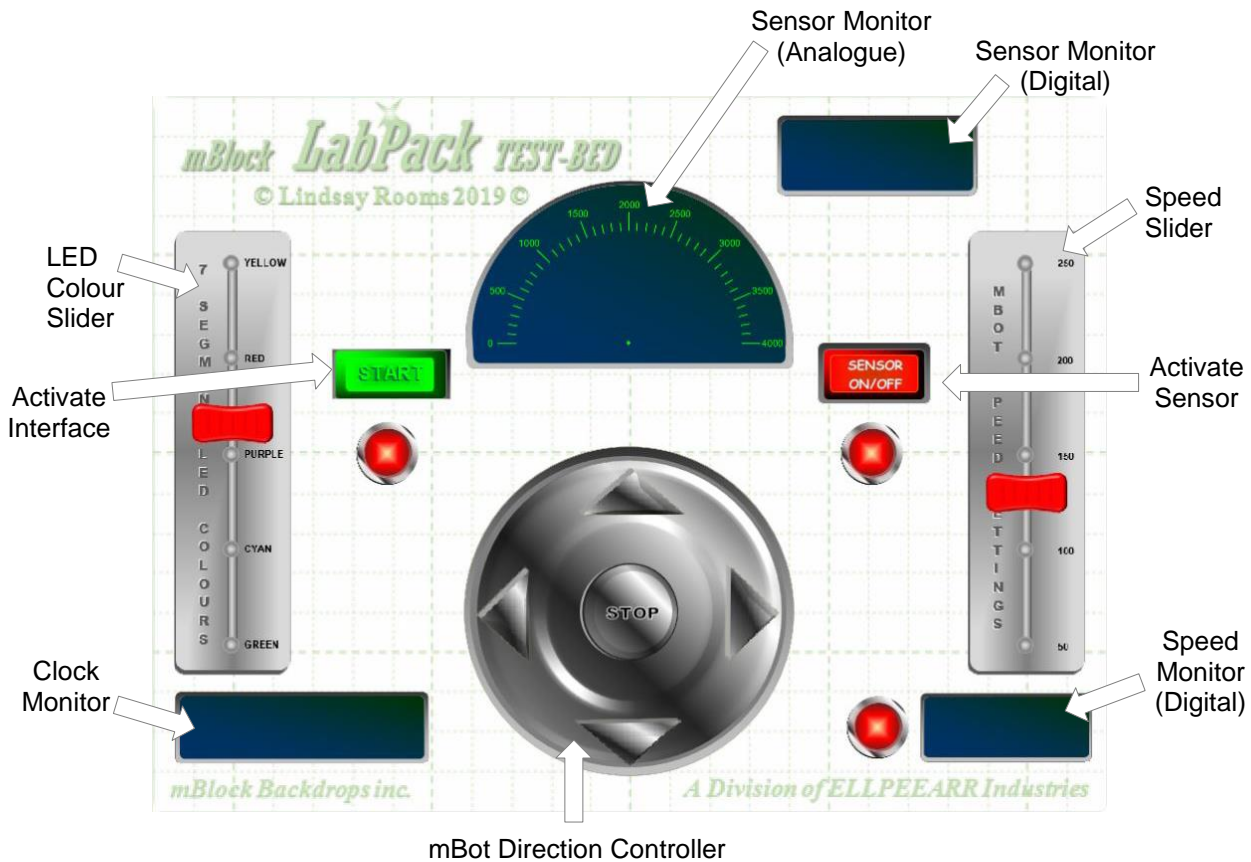
Time to save your project once more.

Step 4: - I used my 'Graph-Paper' backdrop to lay-out the twelve principle sprites that are needed on the interface. This layout is shown at the top of the next page. It's not a bad idea to add these major components now rather than add them one at a time later on as you start to write scripts for them. Doing this now is a bit of a confidence boost too - it looks like you are beginning to have a working project at last. Despite grumbling earlier about the resolution of the Scratch stage, **I cannot fault the way that mBlock 5 handles the resizing of sprite graphics with no apparent lack of quality!**

First, I positioned the backing plate of my mBot Direction Controller (which I named 'Backstop') and the Analogue Sensor Monitor on the centre line of the stage, setting the X & Y coordinates of these to 0,-80 and 0,90 respectively. My graphics also needed to be reduced in size to fit the stage layout; I set the Size of both the backstop and the monitor window to 50. You will need to choose your own Size settings (a percentage of actual image size) to match your own graphics. N.B. The X & Y coordinate positions of all of my sprites (apart from the analogue pointer mentioned earlier) are set to the centre of each graphic.



mBot and Me a Beginner's Guide



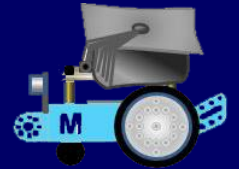
mBot Direction Controller

I positioned my 'LED slider' and my 'Speed Slider' backplate graphics on the horizontal centre-line, setting the X & Y coordinates of these to -200,0 and 200,0 respectively and set the Size setting of each to 90 (I created these just a little bit too large and they do need to remain close to their original size so that the labelling is not distorted). I positioned the On/Off indicator lamp graphic and the Sensor indicator lamp on the horizontal centreline too, setting the X & Y coordinates of these to -122,0 and 122,0 respectively and the Size setting of each to 25. So far, this seemed to be a pleasingly symmetrical layout.

Next, I added the digital sensor monitor window at coordinates 125,150 and set its Size to 28. This was followed by the longer thinner graphic for the digital speed monitor window set at coordinates 190,-138 with a Size setting of 24. The final monitor window was an even longer and thinner graphic for the digital clock window, which was set at coordinates -165,-138 with a Size setting of 50.

Finally, I added the last indicator lamp graphic (to show when mBot is activated) and set its X & Y coordinates to 125,-138 with a Size of 25 and last of all I added my two individual switch graphics above the other two indicator lamps setting the Power On/Off (Start/Stop) switch at -122,40 - Size 60 and the Sensor On/Off switch at 122,40 - Size 50. N.B. We will add the day/date 'engraving' and other descriptive 'engravings' to the interface later! Delete the default 'Panda' sprite and use the 'add new Sprite' button to add your own graphics to create something rather similar to this. Its once again time to save your project file as usual to 'My Projects' in the 'Cloud'.

Its not a bad idea at this stage to make a backup file too, by saving a copy to your computer rather than just to your instant-access 'My Projects' storage area. **Do this backup on a frequent basis too.**



Step 5: - Return to the Devices tab. It's time to add the remaining device scripts that you need here. So far you have created five scripts that you can eventually call using messages to control mBot (you tested these by clicking them in Step 3).

You should already have a script to stop mBot's motors and four directional scripts. Begin by editing these last four scripts, adding the variable 'mBot_Speed' into the little window at the end of each DC motor block - which by default holds a value of 50 (%). Do this to all eight blocks in those four scripts (see the diagram above).

```

when I receive mBot_Forward
  DC motor motor port1 anticlockwise rotates at power mBot_Speed %
  DC motor motor port2 clockwise rotates at power mBot_Speed %
  
```

```

when I receive Poll_Sensor
  forever
    if Sensor_Pos = 1 then
      set Sensor_Value to round ultrasonic sensor port3 distance cm
      broadcast Send_Data
      broadcast Analogue_Pos
    else
      broadcast Zero_Digits
      broadcast Zero_Pointer
  
```

Next you need to make three new scripts to process and then pass data from mBot to Sprites on the stage.

The first of these is shown here on the left. It is actioned when it receives the message 'Poll_Sensor'. Essentially it takes the value that the ultrasonic sensor is outputting at that moment in time and rounds it to a whole number between 0 and approx 400 - the maximum range of the sensor, or therabouts (in centimetres).

Create the whole script as shown, and ignore why most of the other blocks are there for now - all will become clearer as the project develops.

The main thing of interest here perhaps is that the script above calls the next script that you need to make - which is actioned when the 'Send_Data' message is received from that previous script.

This second script is shown here on the right so you need to make this next. The variable 'Sensor_Value' now holds whole-number numeric output from the sensor on mBot.

```

when I receive Send_Data
  set Digits_Shown to length of Sensor_Value
  set Digit_Hundreds to letter Digits_Shown - 2 of Sensor_Value
  set Digit_Tens to letter Digits_Shown - 1 of Sensor_Value
  set Digit_Units to letter Digits_Shown - 0 of Sensor_Value
  broadcast Update_Digits
  
```

The first block in this script 'Digits_Shown' calculates and stores the number of digits that make up that whole-number and the next three blocks then calculate the individual digit that needs to be displayed when the sensor's value is broken down into hundreds, tens and units - simple really. Finally, it broadcasts a message to update the digits displayed on the stage *and this all happens in real-time as the sensor value fluctuates.*



mBot and Me a Beginner's Guide

The third and final addition to the scripts needed here on the 'Devices' tab is shown on the right. You now need to make this too - it resets all of the display digits on the stage to 0 and it then broadcasts the same message as used in the previous script ('Update_Digits'). N.B. Even if some scripts broadcast message names where there is not yet a receiving script to action, this is not a problem. As is now usual at the end of a step sequence, save your project file.



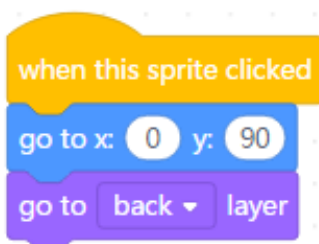
Step 6: - It's time to think about creating scripts for each of your sprites, but first, some notes about the mBlock sprites list in general.

This project requires a rather alarming 49 sprites in total (but 16 of these are text that can be quickly created as needed using the mBlock graphics editor). Of the remainder, several are duplicates; notably the fourteen 7-segment LED digits and the three indicator lamps. You should have already added the first twelve principle project sprites (created in Step 4 earlier) and you should be able to see all of these in the 'Devices', 'Sprites' & 'Background' categories panel on the left of the 'Edit Page' below the 'Stage'.

You can alter the width of the stage area on the mBlock 'Edit Page' and this affects how many sprites are displayed across the 'Devices', 'Sprites' & 'Background' categories panel below the stage. The default width will show three sprite icons across this panel on the majority of desktop monitors, but only one or two sprites wide on most laptop displays. My monitor displays seven rows of sprites without having to scroll down the list; but a disadvantage of having a lot of sprites in a project, is that the new sprite button is always last in the list and any new sprites are added at the bottom of the sprites list too.

Sprites can however be dragged and repositioned in this list very easily and placed in any order, but it makes sense to have the ones you are currently writing scripts for near the top of the list and therefore always visible; so getting used to dragging them into a sequence that suits you is not a bad idea.

One of the two sprites I suggested adding first was the analogue sensor monitor. This in itself is purely a background sprite graphic added to provide realism for a moving pointer rotating in response to feedback from mBot's ultrasonic sensor. As such, this technically does not need any scripts written for it in the 'Scripts' area when it is the currently selected sprite (highlighted in blue with the 'delete-me' cross on its top right corner). **BUT** - it does make sense and good practice to add a script such as the one shown below on the right to EVERY sprite that you create.



This script stores the coordinates that you set when you positioned the sprite accurately on the stage. **x: 0** (exactly on the vertical centre-line) and **y: 90** (90 pixels up from the horizontal centre-line of the stage). The 'hat' block at the top and the 'Motion' block below it clearly indicate that if that sprite is clicked (& inadvertently dragged on the stage) the sprite will always return (go to) its home coordinates position and move behind everything else. When you are experimenting with a project on the edit page then

moving sprites by mistake often happens; but do note that in full-screen 'Presentation Mode' it is not possible to drag sprites (**although you can click them to activate scripts**).



Step 7: - It would be no bad thing at this stage to test the ultrasonic feedback from mBot and then test an analogue pointer on top of your meter graphic (the analogue sensor monitor). Start by switching to the devices tab and clicking the 'Connect' button at the bottom of the left hand panel of the edit page. Choose the 'Sensing' section of the 'Blocks' area and check (tick) the second block choice from the top. This will turn on the following stage monitor:
This shows (if mBot *IS* connected!) the feedback from the sensor - try moving mBot left and right to see the display change in real time.

Click on the 'Variables' section of the 'Blocks' area and do three things.



Check (tick) the boxes next to 'Sensor_Pos' and 'Sensor_Value' and they will then become visible on the stage, each with a value of 0. Below them (still in the 'Variables' section find the 'set (variable name) to (0)' block. So, you don't need to drag this block from here on to the 'Sprites' area. N.B. **It's very useful to know that blocks can be modified and activated whilst they are still in the Blocks area.**

Choose 'Sensor_Pos' as the name inside the block and type '1' to replace '0' in the little window at the end.



To activate it you need to click it somewhere that doesn't action the name or the contents of the window. Try clicking somewhere near the left-hand end of the block over the top of the word 'set'. The block will 'blink' and you will see that the 'Sensor_Pos' monitor window on the stage will change from '0' to '1'.

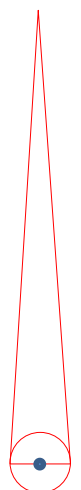
The 'Sensor_Pos' variable is a simple binary '1' or '0' (on/off) switch eventually controlled by the power switch sprite on the interface - for now, you have had to manually provide the '1' in the variable to allow the next test to proceed. Try clicking the hat block at the top of the script you created on the devices tab (this is shown here on the right).



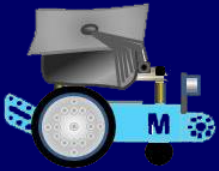
If you look at this script you will realise that it does need 'Sensor_Pos' to equal 1 to proceed. Then it takes the sensor feedback and rounds it to remove the decimal places after the integers and displays this in the 'Sensor_Value' variable. Clicking the hat block should now show whole numbers fluctuating in the 'Sensor_Value' variable monitor on the stage.

Press the red 'stop' button below the stage to halt the script and save your project file.

Step 8: - To see this script fully working, you need to add a new sprite into the project - the meter pointer described earlier on page 100. I drew mine in Word using just a filled red circle and a thin red triangle added over the top of it; but this is such an easy thing to create, so you could just as easily draw this as two vector shapes in mBlock 5's costume editor. (See the diagram on the right). The important thing about the pointer is its x, y position in the editor. It *must* have its pivot point over the centre of the editor screen! Set your own size for this to match the size of your meter graphic. The pointer needs three scripts creating on the sprites tab with the pointer as the active sprite.



These three scripts are shown at the top of the next page.



mBot and Me a Beginner's Guide

```

when this sprite clicked
  go to x: 0 y: 55
  go to front layer

```

```

when I receive Analogue_Pos
  go to x: 0 y: 55
  forever
    point in direction Sensor_Value * 0.45

```

```

when I receive Zero_Pointer
  go to x: 0 y: 55
  point in direction 0
  stop other scripts in sprite
  stop this script

```

Try running the 'Poll_Sensor' message by clicking the 'hat' block on the devices tab again. You should now see the meter pointer fluctuate in response to mBot's sensor. After this test, you can uncheck the variable stage monitors to hide them and also uncheck the 'Sensing' monitor too.

Success, another confidence-boosting milestone - you now have graphical feedback from mBot!

If you have a problem with sprites not being visible despite the 'Show' sprite switch being set to 'on' (e.g. the pointer sprite is behind the monitor sprite) then use the two 'Looks' blocks shown on the right to change the layer (stacking) order of your graphics. Remember this tip - at some point in the making of your project you will certainly need it! Save your project file and back it up again.

```

go to front layer
show

```

Step 9: - The first sprite that I added in step 4 was the backing plate for the mBot Direction Controller (which I named 'Backstop'). This is shown here on the right and is the next thing that you should select so that you can then add the two following scripts to it:



```

when this sprite clicked
  if Switch1_Pos = 0 then
    go to back layer
    go to x: 0 y: -80
  else
    go to back layer
    go to x: 0 y: -80
    play sound Click until done
    change brightness effect by -10
    wait 0.2 seconds
    change brightness effect by 10
    broadcast Stop_Motors

```

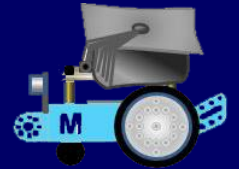
```

when this sprite clicked
  if Switch1_Pos = 0 then
    go to back layer
    go to x: 0 y: -80
  else
    if Switch2_Pos = 0 then
      set Switch2_Pos to 1
      broadcast mBot_Lamp
    else
      set Switch2_Pos to 0
      broadcast mBot_Lamp

```

I decided that the whole backing plate (and not just the 'stop' button in the centre of it) should call the 'Stop_Motors' message on the 'Devices' tab to halt mBot.

This makes sense since the whole area is an easier target to hit when you want to stop mBot in a hurry.



The left-hand script of the two scripts shown at the bottom of the previous page repositions the sprite if necessary (as discussed in step 6 earlier), it then simulates a switch by using the sound effect 'Click' and dimming the graphic briefly (so simple, but very effective). You will need to find a suitable sound file for this (online perhaps where you could search <http://soundbible.com/> for something suitable). Finally, this script broadcasts the message to stop mBot. The second script, on the right of these two scripts essentially switches an indicator lamp from green to red (or vice-versa) to indicate that mBot has stopped or started.

Save your project file.

Step 10: - It makes sense to add the four direction controlling arrow sprites into your project next. Add an up-arrow (for forwards) at the top of the backing plate, a down-arrow (for backwards) at the bottom and left and right-arrows aligned with each other on either side of the backing plate (for turn-left or right). Each of these needs to be positioned precisely and sized to suit, so I set their coordinates as follows:

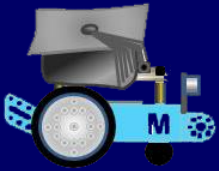


Forwards arrow:	0,	-20	&	size 60
Backwards arrow:	0,	-140	&	size 60
Turn-Left arrow:	-60,	-80	&	size 60
Turn-Right arrow:	60,	-80	&	size 60

Each direction sprite now needs two scripts creating for it; and fortunately, these are nearly identical to the two stop button scripts created in step 9 earlier, so there is a good and very easy method (described below) which will enable you to copy these two scripts into the four direction-arrow sprites.

To do this you need to make the backing plate (stop button) your active sprite by choosing it in the sprites list so that the two scripts you created earlier are visible in the 'Scripts' area. Make sure that you can see your four direction-arrow sprites in the sprites list too by scrolling it up or down it as necessary. Click the left-hand script of the two showing in the 'Scripts' area and drag it to the left. Keep dragging across the 'Blocks' area until your cursor is over the sprites list. As you move across individual sprite icons in the list, any icon under the cursor will rock slightly (or 'wriggle') to indicate that it is ready to receive what you are dragging across. Let go of the dragged script when you are over the forward-direction arrow sprite and it rocks from side-to-side. Click on that sprite to make it active and you should see that the script that you dragged across has now been copied into it. Go back to the backing plate (stop button) sprite and click and drag in the same way the second (right-hand) script across to the forward-direction arrow sprite icon in the sprites list. Make this sprite the active sprite again and you should see (but not very clearly this time) that the second script has also been copied; but has been dropped on top of the first script. All you have to do now is drag the top script sideways slightly to expose the first script and then reposition them neatly side-by-side. Repeat the process of copying the two original scripts by dragging them to each of the other three direction arrow sprites. ***N.B. All of that script copying & repositioning is much quicker to do than it seems in the lengthy description above!***

You should now have the same scripts in both the backing plate (stop button) sprite and in the four direction arrow sprites so you need to go to each of these four in turn and modify the blocks highlighted to match the scripts shown below and on the next two pages. Note that you do need to delete the three 'Looks' blocks 'go to (back) layer' from each of these copied scripts.



mBot and Me

a Beginner's Guide

First, here are the two scripts that you need to modify for the 'move-forwards' arrow sprite:

```
when this sprite clicked
if Switch1_Pos = 0 then
  go to x: 0 y: -20
else
  go to x: 0 y: -20
  play sound Click until done
  change brightness effect by -10
  wait 0.2 seconds
  change brightness effect by 10
  broadcast mBot_Forward

when this sprite clicked
if Switch1_Pos = 0 then
  go to x: 0 y: -20
else
  if Switch2_Pos = 0 then
    broadcast mBot_Lamp
    set Switch2_Pos to 1
  else
    set Switch2_Pos to 0
    broadcast mBot_Lamp
```

Next, here are the two scripts that you need to modify for the 'move-backwards' arrow sprite:

```
when this sprite clicked
if Switch1_Pos = 0 then
  go to x: 0 y: -140
else
  go to x: 0 y: -140
  play sound Click until done
  change brightness effect by -10
  wait 0.2 seconds
  change brightness effect by 10
  broadcast mBot_Backward

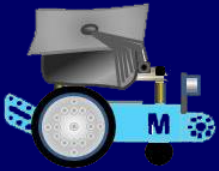
when this sprite clicked
if Switch1_Pos = 0 then
  go to x: 0 y: -140
else
  if Switch2_Pos = 0 then
    broadcast mBot_Lamp
    set Switch2_Pos to 1
  else
    set Switch2_Pos to 0
    broadcast mBot_Lamp
```




Next, the scripts that you need to modify for the 'move-left' arrow sprite:

And finally, the two scripts that you need to modify for the 'move-right' arrow sprite:

Make sure that mBot is still connected. If not, go to the 'Devices' tab and click the 'Connect' button. You can now click your controller arrows on the interface and mBot should respond and change direction with each click.



To make these scripts work you may need to set the 'Sensor_Pos' variable to 1 again (like you did in step 7 to test the 'Poll_Sensor' broadcast) but you should by now be more-than satisfied with your project since you now have both sensor feedback *and* directional control and have demonstrated that two-way communication between a device and a stage interface is possible.

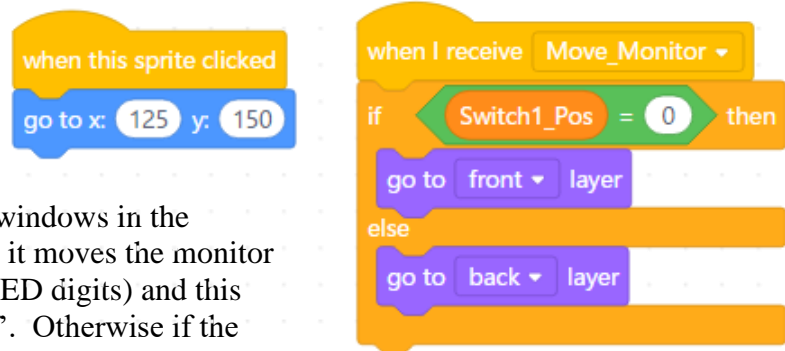
If you have achieved this, then very well done so far. Save your project file and back it up again too.

Step 11: - Buoyed up by the progress of your project so far, it's time to think about digital feedback on the your stage interface. In step 4 I suggested that you added the digital sensor monitor window at the top right side of the interface (at coordinates x: 125,y: 150).

Click on this sprite in the sprites list and (if you haven't done it already) name it 'Sensor_Monitor'.

This monitor sprite does need the two scripts (as shown here on the right) added to it. The first of these should by now need no further explanation, whilst the second is fairly easy to understand.

If it receives (as will all the other monitor windows in the project) the message 'Move_Monitor' then it moves the monitor graphic to the front layer (in front of any LED digits) and this suggests that 'power' has been turned 'Off'. Otherwise if the power switch is 'On', then the monitor window is moved to the back layer and the LED digits are once more visible (suggesting 'powered-up'!).



It is here that you now need to add hundreds, tens & units digits to represent the LED feedback (in centimetres)from mBots ultrasonic sensor.

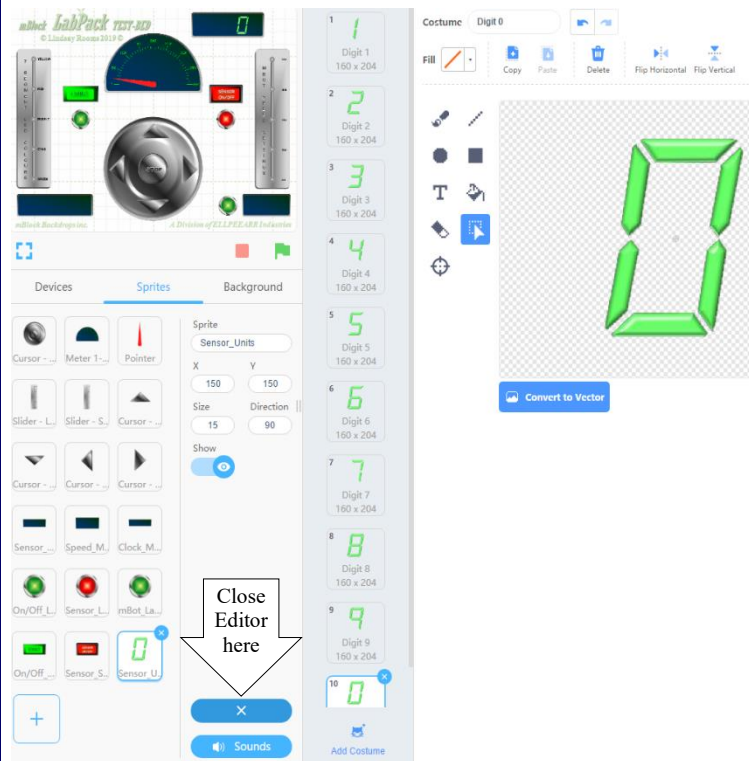
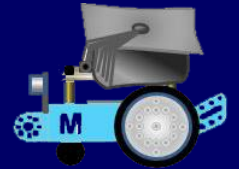
N.B. You have already written a script on the 'Devices' tab to broadcast this data.

You now need to create a new sprite with ten individual costume graphics (1 to 9 and 0) so click on the new sprite button at the bottom of the sprites list to enter the 'Costumes' library'. If necessary, upload all of your digit graphics into the 'My Costumes' library first and then open from that library the number '1' 7-segment LED (.png) file that you created using my instructions in Chapter 14 on pages 90 to 94). Name this sprite 'Sensor_Units' and set its coordinates to x:150, y:150 and size: 15.

You should now see your LED digit nicely positioned in the monitor window at the top of the interface.

Click on the 'Costumes' button at the bottom of the 'Edit Page' (next to the sprites list) to open the costume editor and name this first costume as 'Digit_1'. The grey strip between the costume editor window and the stage / sprites list panel is the panel containing the list of costumes available to the sprite currently selected.

At the top of this panel you will be able to see your 'Digit_1' graphic (and this will also be displayed in the editor window too). At bottom of the costumes panel is a little blue icon of a cat's head and the words 'Add Costume'. Click here and it will take you back to the costume libraries.



Open your number '2' graphic from the 'My Costumes' library and name it as 'Digit_2'. Note that this will become the active costume and both the editor and the monitor on the stage will display the active sprite. Add and rename the remaining graphics until you have the sequence of costumes (1 to 9 and 0). You can see both the costume editor window, the sprites list and the costume list for the 'Sensor_Units' sprite here on the left. It is rather gratifying to click up-and-down the costume list watching the digit in the monitor window change as you do so.

Close the costume editor, using the (X) button that replaced the 'Costumes' button at the bottom of the 'Edit Page'. Add the two scripts shown here (below right) to your new 'Sensor_Units' sprite. The first script positions the sprite; but rather than a value being inserted into the block does so by

using data stored in variables 'Digit_Xpos' & 'Digit_Ypos'.

This is quite a good ploy to get used to here, since you can use another script to alter these variables to reposition the 'Sensor_Units' digit (this is good practice in other projects!).

The decision making part of this script says that if the variable 'Digit_Units' holds the value '1' then choose costume '1' whilst if 'Digit_Units' contains '2' then choose costume '2' and so on.

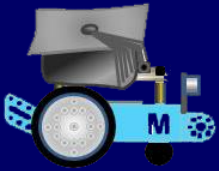
However, costume '10' of this sprite contains '0'; so, the first bit of the script makes a decision to switch to that costume if 'Digit_Units' holds a value of less than '1'; otherwise it continues to choose the costume matching its value!

```

when I receive Update_Digits
  go to x: Digit_Xpos y: Digit_Ypos
  if Digit_Units < 1 then
    switch costume to Digit (italic) 0
  else
    switch costume to Digit_Units
  show
  
```

```

when I receive Change_LED
  if Slider1_Switch_pos = 5 then
    set color effect to 165
  if Slider1_Switch_pos = 4 then
    set color effect to 125
  if Slider1_Switch_pos = 3 then
    set color effect to 95
  if Slider1_Switch_pos = 2 then
    set color effect to 40
  if Slider1_Switch_pos = 1 then
    set color effect to 0
  
```

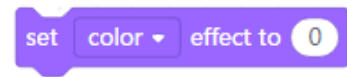


The second script shown on the previous page is made up of five simple *'if/then'* decisions which reflect the position of the slider button on the LED colour backplate.

N.B. Once you have added the first of these five *'if/then'* blocks (and added the completed green *'Operators'* block and the purple *'Looks'* block to it, you can duplicate it four more times and change the colour effect number and the switch value number for each *'If'* decision in turn.

A switch value of 5 sets the LED to yellow, a value of 4 to Red, a value of 3 to Purple, a value of 2 to Cyan and a value of 1 returns the colour change to 0 (reverting the LED to its original colour, Green).

Without the slider switch values being set you can't see the results of these changes by testing this script yet; but, if you want to see what they look like, then find the *'set (colour) effect to ()'* block in the *'Looks'* menu, type in one of the values above and click the left-hand end of the block (over the word *'set'*) and you will see your 7-segment LED graphic in its monitor window on the stage change colour.



That's quite a lot of work that you have just added, so it makes sense to save your project file once again.

Step 12: - You now need to add the remaining two digit sprites; *'Sensor_Tens'* and *'Sensor_Hundreds'*. This is not nearly as much work as in step 11 above, since both of these are essentially duplicates of the *'Sensor_Units'* sprite that you have just made - and best of all, remember that duplicated sprites contain the same scripts as their parent sprite!

Go to your *'Sensor_Units'* sprite in the sprites list and right-click it. You are given the choice of *'duplicate'* or *'delete'* so click on ***'duplicate'*** and avoid clicking on *'delete'*! You will see a new sprite (added to the bottom of the sprites list) labelled *'Sensor_Units2'* - rename this as *'Sensor_Tens'*.

Duplicate this once again and label it as *'Sensor_Hundreds'*. You might at this stage do as I did and reposition the order of these digit sprites in the sprites list, sensibly putting them into the standard mathematical sequence of hundreds, tens and units.

Each of these sprites will have the ten individual costume graphics (1 to 9 and 0) as in the parent sprite and they will also contain the two scripts that you created for the parent sprite in step 11.

The very good news is that the *'hat'* block *'when I receive (Change_LED)'* script is identical for all three of your digit sprites; and since you do want them all to change colour, so there is nothing to change here.

Both of the *'hat'* block *'when I receive (Update_Digits)'* scripts in your two duplicated sprite scripts need to be marginally altered to reposition these new digits to the left of the *'units'* digit and to choose the costume matching the tens and hundreds variables.

To do this, go to each of these new scripts in turn and in the two places where each of these scripts refer to the *'Digit_Units'* variable replace them with the *'Digit_Tens'* and *'Digit_Hundreds'* variables as appropriate for each script.

Digit_Units

See the contents of these two scripts on the next page.



```

when I receive Update_Digits
go to x: Digits_Xpos - Digits_Space y: Digits_Ypos
if Digit_Tens < 1 then
switch costume to Digit (italic) 0
else
switch costume to Digit_Tens
show
    
```

The modified script for the 'Sensor_Tens' sprite is shown here on the left.

The 'go to x: () y: ()' block needs to be modified as shown on the left to include an 'Operator' block '() minus ()' added into the x: window which shows the 'Digits_Xpos' variable first with a variable called 'Digits_Space' being deducted from it.

The resultant x position is therefore less than it was for the x position of the 'Sensor_Units' digit and moves the 'Sensor_Tens' digit a pre-set distance to the left. In the script shown below (for the 'Sensor_Hundreds' sprite) then this value is doubled (multiplied by 2) to position that digit using twice the value of 'Digit_Space' moving it to the left and leaving room for the 'tens' digit in the middle).

This all ties in with a principle that you can use in other projects; this good practice concept was hinted at in step 11 earlier and suggested that using data stored in the variables 'Digit_Xpos' & 'Digit_Ypos' (and also in the value of the variable 'Digit_Space') rather than a value being inserted directly into the

```

when I receive Update_Digits
go to x: Digits_Xpos - Digits_Space * 2 y: Digits_Ypos
if Digit_Hundreds < 1 then
switch costume to Digit (italic) 0
else
switch costume to Digit_Hundreds
show
    
```

block will allow another script to alter these variables and thus reposition these digits as required. This time, the block modification requires a (minus) 'Operator' block inside the first window of a (multiply) 'Operator' block.

Since these variables have not yet been set, you might want to manually position and size these last two sprites. Set the 'Tens' sprite to position 125,150 and the 'Hundreds' sprite to 100,150.

You now need to return to your 'Sensor_Units' sprite and add another little script that can set these variables. Remember that it is sensible to always create a sprite script enabling it to reposition itself (as discussed in step 6 earlier).

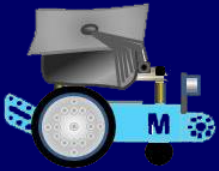
In this case this is a rather different, but very valuable method.

To enable the 'Sensor_Tens' and 'Sensor_Hundreds' digits to automatically position themselves to the left of the 'Sensor_Units' digit you need to set the values of the variables 'Digit_Xpos', 'Digit_Ypos' & 'Digit_Space'. This creates a three digit number that can be easily repositioned - move the units digit, and they all move! You may also find this concept of considerable value in other projects.

Go to 'My Blocks' in the blocks list and create a new block called 'Digit_Pos'. Then use three Variables 'set' blocks to set the positional values as shown in the diagram on the right (you might want to experiment with your own 'Digit_Space' value here).

```

define Digit_Pos
set Digits_Xpos to 150
set Digits_Ypos to 150
set Digits_Space to 25
    
```



mBot and Me a Beginner's Guide

```

when I receive Update_Digits
  Digit_Pos
  go to x: Digits_Xpos y: Digits_Ypos

```

Finally, you need to modify the 'hat' block 'when I receive (Update_Digits)' script (as shown here on the left) by adding your newly defined 'Digit_Pos' block to it. Drag the block across and position it at the top of the script between the 'hat' block and the 'go to x: () y: ()' block.

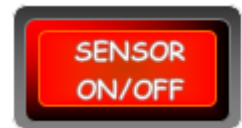
To test that these scripts work, go to the 'Events' blocks menu and find the 'broadcast (message)' block.

Choose 'Poll_Sensor' in the messages list and then click the left-hand end of the block to broadcast the message. You should now see your three digits react to match the sensor feedback from mBot. You will also see your analogue pointer fluctuate to (just like it did when you first tested it in step 8).

If this doesn't seem to work, check the 'Devices' tab scripts that you created in step 5 carefully (particularly the 'Send_Data' script that breaks up the 'Sensor_Value' into the individual digits that you are trying to display here).

Save and backup your project file once more.

Step 13: - It's time to programme the right-hand of the two switches on the interface that we added in step 4. You could use just about any graphic of your own for this. I named this graphic 'Sensor_On/Off' and it only has one costume, relying on using the 'change (brightness)' block (demonstrated in step 9) to give an impression of interaction by dimming it briefly and making a 'click' sound whenever it is clicked.



This sprite has only one costume, but it still toggles 'On' & 'Off' in the same way as a two-costume (moving/changing graphic) sprite would. The 'when this sprite clicked' script shown on the next page essentially sets an 'On' position (1) for a first click and an 'Off' position (0) after a second click of the same sprite graphic. The 'Sensor_Pos' variable enables the 'Sensor_Lamp' sprite to toggle between its two costumes, showing a red lamp for 'Off' and a green lamp for 'On'.

There are in fact two scripts, the second one being a good use of a self-defined (or custom) 'My Block'. Since the same bit of scripting (added under the 'My Block' 'hat' called 'Operate_Switch') is used twice, it makes sense to only write this once and then just add the single 'Operate_Switch' block into either half of the main script). This shortens what would have been a long script to see without scrolling it up and down.

```

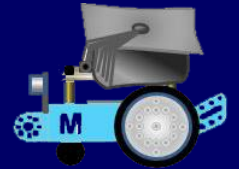
when this sprite clicked
  go to x: 123 y: 40
  if Sensor_Pos = 1 then
    Operate_Switch
    set Sensor_Pos to 0
  else
    Operate_Switch
    broadcast Zero_Digits
    set Sensor_Pos to 1

```

```

define Operate_Switch
  play sound Click until done
  change brightness effect by -10
  wait 0.25 seconds
  change brightness effect by 10
  broadcast Sensor_Lamp
  broadcast Poll_Sensor

```



When you try test-clicking the switch sprite on the interface for the first time you should see (if mBot is connected, that is!) the digital feedback revert to 000 and the analogue pointer drop to 0 when the switch is 'Off' and then real-time sensor data displayed on both when the switch is clicked to 'On'.

Save your project file.

Step 14: - The aforementioned 'Sensor_Lamp' sprite needs to be programmed next to complete this part of the interface involving mBot's ultrasonic sensor. This is a very simple and speedy modification, as long as you have the costumes for it ready-made and uploaded into your 'My Sprites' library.

The 'Sensor_Lamp' sprite needs to have two costumes; costume 1 is a red lamp graphic and costume 2 an identical lamp but with the centre ('bulb' part) green. Chapter 14 (page 95) gives a clue in how to draw these in Word.

The script attached to this sprite (shown here on the right) is very straight-forward and easy to understand. If the switch has a value of '1' ('On') then show the green lamp and if the switch is anything else, then show the red lamp ('Off').

```

when I receive Sensor_Lamp
  go to x: 122 y: 0
  if Sensor_Pos = 1 then
    switch costume to Lamp_Green
  else
    switch costume to Lamp_Red
  
```

Try clicking your 'Sensor_On/Off' button sprite on the interface once again and as well as the sensor feedback being turned on or off you should also see the Lamp changing from green to red and red to green each time that you click it.

Save and backup your project file once more.

```

when this sprite clicked
  go to x: -120 y: 40
  if Switch1_Pos = 0 then
    Operate_Switch
    switch costume to Stop
    broadcast Power_Lamp
    set Switch1_Pos to 1
    Start_Clock
  else
    Operate_Switch
    switch costume to Start
    set Switch1_Pos to 0
  
```

```

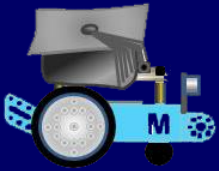
define Operate_Switch
  broadcast Power_Lamp
  broadcast Move_Monitor
  start sound Click

define Start_Clock
  
```

Step 15: - It probably makes sense to next undertake the programming for the other (left-hand) switch on the interface, the 'Power_On/Off' switch.

This is a two-costume (moving/changing graphic) sprite which sets an 'On' position (1) for the first costume and an 'Off' position (0) for the second costume of the sprite.

There are initially two scripts that need to be created here and these are very similar to the scripts created for the right-hand switch sprite detailed in step 13 previously. These are shown here on the left.



You need to make this second switch script next. This is once again a self-defined 'My Block' also named like its predecessor, 'Operate_Switch'.

There will eventually be a third 'My Block' script 'Start_Clock' added to this sprite - but other than adding the block name as shown in the diagram at the bottom of the previous page, not just yet. **This additional script will be fully completed when the digital clock is eventually created to finalise the interface.**

When this 'power' switch is in the 'On' position it allows all the other button sprites to work and eventually will also activate the 'Start_Clock' display routine. When this switch is in the 'Off' position it stops the clock and deactivates all of the other button sprites.

Most importantly this script also broadcasts the 'Move_Monitor' message which repositions the three monitor window graphics to the front layer (in front of the three LED displays) to simulate the effect of the LEDs being turned off.

Step 16: - To enable all of the monitor windows on the interface to be moved to the front as described above, they each need the same script attached to them. This script is actioned when the 'Move_Monitor' message is received.

Go to your 'Sensor_Monitor' sprite and create the script shown here on the right then drag this script across into the sprites list to duplicate it into your both your 'Speed_Monitor' sprite and your 'Clock_Monitor' sprite.

```
when I receive Move_Monitor
if Switch1_Pos = 0 then
  go to front layer
else
  go to back layer
```

Now when you test-click your 'Power_On/Off' switch on the interface you should see any digits in the sensor monitor window being hidden and then reappearing when the switch is clicked once again as well as the costume of the switch sprite changing from the 'Start' button to the 'Stop' button.

It makes sense to also write the usual repositioning script for all three monitor windows, so return to the 'Sensor_Monitor' sprite and create the script shown on the right. Drag (as you have done before) to duplicate this on both your 'Speed_Monitor' sprite and your 'Clock_Monitor' sprite and adjust the x: and y: coordinates accordingly for each of them.

```
when this sprite clicked
  go to x: 125 y: 150
```

```
when I receive Power_Lamp
  go to x: -122 y: 0
if Switch1_Pos = 1 then
  switch costume to Lamp_Green
else
  switch costume to Lamp_Red
```

Step 17: - To complete the effect of the switch that you programmed in step 15 earlier, it now needs the 'Power_Lamp' sprite positioned below it to be programmed.

Create the script shown here on the left and then test-click the 'Power_On/Off' button on the interface again - all should work as before with the addition of the lamp being switched from red to green and vice-versa. You should be buoyed-up with your successes once again, so save and backup your project file.



Step 18: - Sliders are a quick way of altering the values stored in a variable and are so useful, that one of the three monitor windows that can be made visible on the stage to show the contents of a variable is the horizontal slider shown here on the right. The slider button can be dragged either left or right to anywhere on its track and the value can be set (in incremental values of 1) from 0 on the left to 100 on the right.



Sliders such as this are hard to emulate well in an mBlock stage graphic for several reasons.

The first reason being the fact that although sprites on the stage can be dragged, moved and clicked when you are working in the 'Edit' page, they cannot be moved in full-screen 'Presentation Mode' where only clicking a sprite is possible. This rules out dragging a slider button up and down (or across) a slider track.

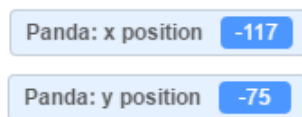
Secondly, you need a way of analysing exactly where the slider button is on its track and in the example above, you would need to create 100 rather complex decision making scripts to work out the value indicated by a slider buttons position.

Thirdly, it's a complex decision to decide where to position a slider on the stage.

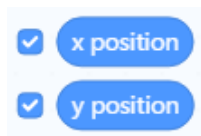
Remember, the zero position for the x and y coordinates of any sprite is in the centre of the stage; therefore, only sprites in the upper right quadrant of the stage have positive integers and calculating with negative numbers is difficult.

In the lower left quadrant of the stage, both x and y coordinates are negative integers whilst in the other two quadrants, one of the coordinates (x or y) will be a negative integer.

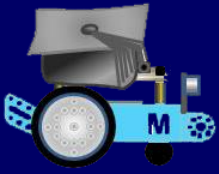
For the two vertical sliders that I had in mind for this interface (a LED Colour Changer and a Speed Changer) I decided that there only needed to be five stopping places for the button on each slider e.g. in the 0 to 100 scenario described earlier this only need a calculation for every 25 points 0,25,50,75 & 100 and not one calculation for each of 100 stopping points!



After much deliberation and experimentation, I found that in the 'Motion' blocks menu there were two 'Reporter' blocks labelled 'x position' and 'y position' and if you ticked the checkbox next to them then this turned on labelled monitor windows on the stage showing the position of the current sprite; and if you dragged the sprite across the stage then these coordinates changed. What, I speculated - could the y position of a sprite be copied into another variable using the 'Reporter' block value at that moment in time whenever that sprite was clicked, and that value be used make a decision?



Yes, this was the key to solving the problem and if the sprite in question was the slider's backplate and not the button then a range of values 0 to 25, 25 to 50, 50 to 75 and 75 to 100 could be used. If the backplate sprite was clicked somewhere in the 50 to 75 range for example (an actual y pos. click = 68) then the y position could be told to accurately jump to the next higher value (75) and the button sprite could move to that position too. Problem basically solved! The button on the slider needs to be just a graphical adornment - not actually working but just pretending to be the operating bit of the slider.



mBot and Me a Beginner's Guide

```
define Make_Slider_Choice
if Slider1_Ypos > 250 and Slider1_Ypos < 320 then
set Slider1_Ypos to 80
set Slider1_Switch_Pos to 5
if Slider1_Ypos > 200 and Slider1_Ypos < 250 then
set Slider1_Ypos to 50
set Slider1_Switch_Pos to 4
if Slider1_Ypos > 150 and Slider1_Ypos < 200 then
set Slider1_Ypos to 0
set Slider1_Switch_Pos to 3
if Slider1_Ypos > 100 and Slider1_Ypos < 150 then
set Slider1_Ypos to -50
set Slider1_Switch_Pos to 2
if Slider1_Ypos > 80 and Slider1_Ypos < 150 then
set Slider1_Ypos to -100
set Slider1_Switch_Pos to 1
if Slider1_Ypos = 80 then
set Slider1_Ypos to 100
broadcast Move_Slider1
```

The long 'My Blocks' script shown here on the left uses this calculation concept for the 'LED Slider' backplate (which is positioned on the left-hand side of the interface) - an image of this is shown here on the right.



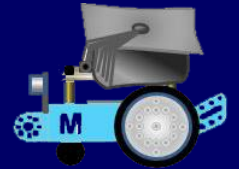
In addition to the 'My Blocks' self-defined block script called 'Make_Slider_Choice' shown here on the left, a second script (shown at the top of the next page) was also created to activate the 'Make_Slider_Choice' script.

The first bit in this 'when this sprite clicked' script checks to see if the 'power' switch has been activated and if not just repositions the sprite and makes sure that it is at the back (& therefore always behind the slider button).

The main part of this script is the 'else' decision part which only works when the power switch is 'On'. The crucial bit here is setting the slider y pos. variable to the y position of the mouse pointer at the moment when it was clicked (exactly as outlined in the basic concept outlined on the previous page).

There is a new addition to this script. It uses an 'Operators' block to add the value contained in a variable called 'Plus_Factor' to the value of the y pos. variable - but more about this shortly.

The last bit of this script repositions the sprite if necessary and then calls the self-defined block 'Make_Slider_Choice' script shown above.



```

when this sprite clicked
if Switch1_Pos = 0 then
  go to x: -200 y: 0
  go to back layer
else
  set Plus_Factor to 200
  set Slider_Xpos to -200
  go to mouse-pointer
  set Slider1_Ypos to y position + Plus_Factor
  go to x: -200 y: 0
  go to back layer
  Make_Slider_Choice
  
```

On the left is the 'when this sprite clicked' script discussed at the bottom of the previous page which calls the 'Make_Slider_Choice' script (also shown and described on the previous page).

The 'Make_Slider_Choice' script is made up of five separate 'If' decisions one for each area of the slider (but there is also a sixth decision at the end - more about that later too).

Each decision determines (from the top of the slider downwards) if the captured y pos. click of the mouse on the sprite is in between a range of figures and if this is true then it sets a specific new value for the y pos. variable and sets a specific value on which the slider button can be positioned.

If the captured y pos. click of the mouse on the sprite is NOT in between the range of figures specified, then it is ignored and the action jumps to the next 'If' decision

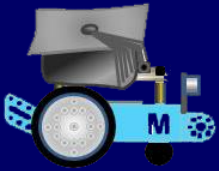
checking the next range of figures down the slider sprite. This is repeated until the fifth if decision is reached and checked.

One of these five 'If' decisions will have set the new value for the y pos. variable and a specific value for the slider button position. So finally, the bottom block will be activated to broadcast the message to move the slider button.

Why is there a sixth 'If' decision at the bottom (before the broadcast block)? - This decision looks at the slider value variable and if it contains 80 it bumps it up to 100 (and is otherwise ignored if the variable contains any other value). This was a 'workaround' I had to invent to correct a problem with the topmost 'If' decision.

When this decision is true, it sets the slider value to 80 (but this really needs to be set to 100 to position for the button correctly) *BUT* if 100 was the value returned by this script decision then the slider would just not work - but why not? I guess that my Scratch programming prowess was just not good enough to solve this. However, checking the value and changing it again using the sixth 'If' decision worked - and I have no idea why!

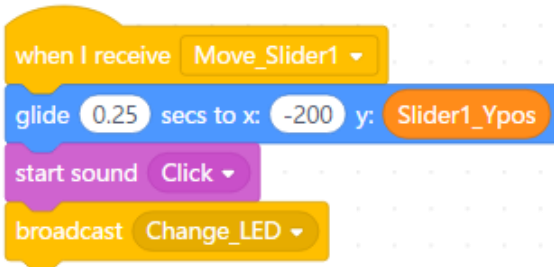
Why the need for adding a 'Plus_Factor' value of 200 to the y pos. variable? - Setting the 'Plus_Factor' to 200 and then adding it to every y position turns all negative integers into positive ones which are much easier to calculate. (-180 is the lowest y position on the stage) so setting the 'Plus_Factor' to 180 would work to create positive values; but a round value of 200 makes the values much easier to understand.



mBot and Me a Beginner's Guide

If you understood all of that, you can now write the 'when this sprite clicked' script and the 'My Blocks' self-defined block 'Make_Slider_Choice' script for your LED_Slider sprite.

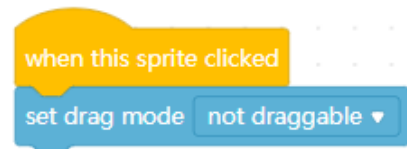
You can then carry on by making the next very short script too. Creating the 'when I receive message' script for the 'LED_Slider_Button' (shown here on the right) is by comparison very simple indeed.



This script makes very good use of the 'Motion' block 'glide to x: () y: ()'. The y position uses the value contained in the slider y pos. variable.

If you do click and drag the 'LED_Slider_Button' by mistake, it can move out of its slot-line until you next click on the slider backplate to reposition it.

In this case (because the button *has* to move when instructed by the script above) you cannot use the usual sprite repositioning trick that you have used for all other sprites. All that you have to do is to add the simple one-block script (shown here on the right) to the 'LED_Slider_Button' sprite and this will stop it from being dragged.



When I tested this project on my surface laptop (which has a touch screen) there was a totally unexpected but massive bonus! I tried operating the sliders with a finger on the screen and not using a mouse and if you swipe, push or pull the slider button with a finger it really looks as though you are moving the slider button - but in reality, your finger position when you remove it from the screen is the last known y pos. and the button glides to that position!

In step 11 you created a script called 'Change_LED' and added it to each of the three (hundreds, tens & units) 7-segment LED digit sprites that you made to provide digital feedback from mBots sensor.

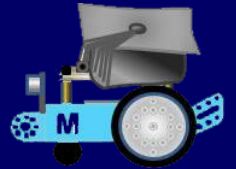
This script uses the value of the 'Slider1_Switch_pos' variable that is set by the 'Make_Slider_Choice' self-defined block script that you have just created.

A 'Slider1_Switch_pos' variable value of 5 sets colour effect of the LEDs to yellow, a value of 4 to Red, a value of 3 to Purple, a value of 2 to Cyan and a value of 1 returns the colour effect to 0 (reverting each LED to its original colour, Green).

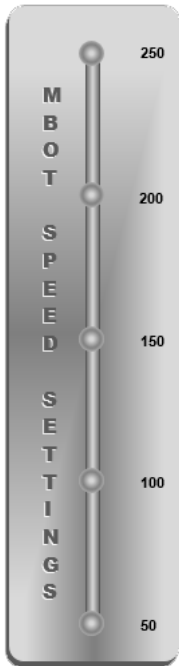
If you try clicking in the zone below each of the colour names on the slider backplate then the 'LED_Slider' button will move to stop at that colour name and the colour of the three LED digits in your sensor monitor window will change too.

Remember that if you click and drag the 'LED_Slider_Button' nothing will happen!

As usual, this is a good time to save and backup your project.



Step 19: -



Now that you have completed the previous step and have created one working slider (and understand the principle of how it works) you can programme the second one which will enable you to adjust mBots motor speeds easily.

I named my Speed Settings Slider backplate as 'Speed_Slider'.

You need to copy the two scripts from your LED_Slider across to this sprite by using the dragging-across method described in step 10 earlier. They only need a few modifications for them to work with your second slider.

```

when this sprite clicked
  if Switch1_Pos = 0 then
    go to x: 200 y: 0
    go to back layer
  else
    set Plus_Factor to 200
    set Slider_Xpos to 200
    go to mouse-pointer
    set Slider2_Ypos to y position + Plus_Factor
    go to x: 200 y: 0
    go to back layer
  Make_Slider_Choice
  
```

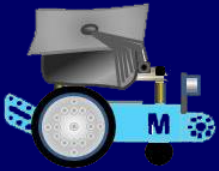
The 'when this sprite clicked' script is easy (see above) there are only 4 things to modify. In the two blue 'go to x: () y: ()' blocks, change the value of x in each from -200 to 200 and then change the value of the orange 'set (Slider_Xpos)' block from -200 to 200.

Change the variable name in the orange block below the blue 'go to (mouse-pointer)' block where it says 'set (Slider1_Ypos)' and change this to the (Slider2_Ypos) variable name. In the 'Make_Slider_Choice' self-defined block script, there are more changes, but these are even easier to make (this script is shown on the next page).

In the six 'If' decisions in this script there are eleven orange 'reporter' blocks which report the contents of the 'Slider1_Ypos' variable. Delete all of these and replace them with new orange 'reporter' blocks reporting the contents of the 'Slider2_Ypos' variable.

At the very end of the six 'If' decisions in this script is a broadcast block calling the 'Move_Slider1' message. Change this to call the 'Move_Slider2' message instead and the scripts for the 'Speed_Slider' backplate sprite are now complete.

As usual, save and backup your project.



```

define Make_Slider_Choice
if Slider2_Ypos > 250 and Slider2_Ypos < 320 then
  set Slider2_Ypos to 80
  set Slider2_Switch_Pos to 5
if Slider2_Ypos > 200 and Slider2_Ypos < 250 then
  set Slider2_Ypos to 50
  set Slider2_Switch_Pos to 4
if Slider2_Ypos > 150 and Slider2_Ypos < 200 then
  set Slider2_Ypos to 0
  set Slider2_Switch_Pos to 3
if Slider2_Ypos > 100 and Slider2_Ypos < 150 then
  set Slider2_Ypos to -50
  set Slider2_Switch_Pos to 2
if Slider2_Ypos > 80 and Slider2_Ypos < 150 then
  set Slider2_Ypos to -100
  set Slider2_Switch_Pos to 1
if Slider2_Ypos = 80 then
  set Slider2_Ypos to 100
  set Slider2_Switch_Pos to 5
broadcast Move_Slider2

```

On the left is the 'Make_Slider_Choice' self-defined block script for the 'Speed_Slider' (as described on the previous page).

Creating the 'when I receive message' script for the 'Speed_Slider_Button' is slightly more complex than the script for the 'LED_Slider_Button' described earlier.

The script to move the 'Speed_Slider_Button' is shown at the top of the next page and once again uses a 'glide' block and a 'sound' block (in exactly the same way that the script that you created for your 'LED_Slider_Button' did).

However, the new 'Move_Slider2' script also needs five 'If' decisions added to it to (for each of the five stop positions on the slider) to set individual speeds for mBot.

After creating it try clicking in the zone below each of the speed settings on the slider backplate.

The 'Speed_Slider' button will move to that speed-stop setting.

Add to the 'Speed_Slider_Button' the same one-block script that you used before to prevent it from being dragged by mistake.

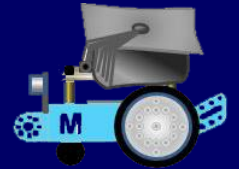
```

when this sprite clicked
  set drag mode not draggable

```

Connect mBot and try it out.

It's time to save and backup your project once again too.



The 'Speed_Slider_Button' script described on the previous page is shown below:

```

when I receive Move_Slider2
  glide 0.25 secs to x: 200 y: Slider2_Ypos
  start sound Click
  set mBot_Speed to 0
  if Slider2_Switch_Pos = 5 then
    set mBot_Speed to 250
  if Slider2_Switch_Pos = 4 then
    set mBot_Speed to 200
  if Slider2_Switch_Pos = 3 then
    set mBot_Speed to 150
  if Slider2_Switch_Pos = 2 then
    set mBot_Speed to 100
  if Slider2_Switch_Pos = 1 then
    set mBot_Speed to 50
  broadcast Update_Speed
  
```

Finally, this step would not be complete without some useful facts about Makeblock's motors and speeds:

In mBlock 3, mBot's motors needed a minimum speed setting of 70 / 255 to drive devices without labouring. In mBlock 5 the power setting now uses percentages - a speed setting of 70 / 255 in mBlock 3) = 27.5% in mBlock 5.



Speed '70' does not however reflect the real running speed value.

mBot's *Real Running Speed* is dependent on both voltage and speed values and the calculation for this =

$$(\text{set speed} / \text{max speed}) \times (\text{battery voltage} / \text{motor voltage}) \times \text{motor no-load speed}$$

The power supplied through mBots battery is 3.7V (fully charged) and the DC motors have a rated voltage of 6V (but individual motors do vary a bit in actual power output).

mBot motors have a gear ratio of 1:48 and run at a no-load speed of 200 RPM ($\pm 10\%$).

The max. speed of these motors is 255.

At speed **100 (40%)** mBots *Real Running Speed* = $0.392 \times 0.617 \times 200 = 48 \text{ rpm}$.

At speed **70 (27.5%)** mBots *Real Running Speed* = $0.275 \times 0.617 \times 200 = 34 \text{ rpm}$.

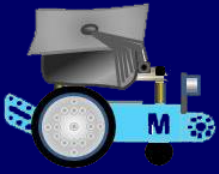
At speed **50 (20%)** mBots *Real Running Speed* = $0.1961 \times 0.617 \times 200 = 24 \text{ rpm}$.

A speed setting of 50 is about 20% power, but to drive mBots motors slowly but adequately 30% power is rather more realistic; and a speed setting of 100 (which equates to 40%) is a very good speed for mBot control.

My own mBot veers to the right very slightly: - So to correct this I experimented with adding 2% of extra power to the right-hand motor (connected to motor port M2) - this script modification worked very well.

```

when I receive mBot_Forward
  DC motor motor port1 anticlockwise rotates at power mBot_Speed %
  DC motor motor port2 clockwise rotates at power mBot_Speed + 2 %
  
```



mBot and Me a Beginner's Guide

Step 20: - The last block added to the *'Speed_Slider_Button'* script shown on the previous page is a broadcast block. The message it is broadcasting is *'Update_Speed'*. This has not been written yet and needs to be added to three new LED digit sprites which need to be created and displayed in the speed monitor window below the Speed Settings Slider.

You already have three LED digit sprites named *'Sensor_Hundreds'*, *'Sensor_Tens'* & *'Sensor_Units'* which you can duplicate and rename the duplicates as *'Speed_Hundreds'*, *'Speed_Tens'* & *'Speed_Units'*. You have duplicated sprites before, so go to the sprite pane on the left of the edit page and right-click on each of your sensor LED sprites in turn and choose 'duplicate' from the drop down list. Name each new sprite as described above.

Each of these new sprites will have the scripts from its parent sprite attached. In each of your new 'Speed' sprites you need to keep the *'when I receive (Change_LED)'* script so that the colour of each of these new LEDs can still be changed with the colour slider.

Delete any other scripts in each of these new 'Speed' sprites in readiness to create the three new *'when I receive (Update_Speed)'* scripts mentioned above.

```
when I receive Update_Speed
  go to x: 215 - Digits_Space y: -138
  if mBot_Speed = 250 then
    switch costume to Digit 5
  if mBot_Speed = 200 then
    switch costume to Digit 0
  if mBot_Speed = 150 then
    switch costume to Digit 5
  if mBot_Speed = 100 then
    switch costume to Digit 0
  if mBot_Speed = 50 then
    switch costume to Digit 5
  show
```

Start with your new *'Speed_Units'* sprite and create the short script shown on the right. Note that there are no decisions to be made on this sprite since all of the speed settings end in zero - so the script just positions costume *'Digit 0'* accurately in the monitor window.

```
when I receive Update_Speed
  go to x: 215 y: -138
  switch costume to Digit 0
  show
```

Next you need to modify the programming of your *'Speed_Tens'* sprite.

In addition to the *'when I receive (Change_LED)'* script that you left behind; you now need to create the decision making script shown on the left which will set the middle ('tens') digit on the display to the required costume for each speed setting of the slider.

N.B. These costumes only need to display either '5' or '0'.

Finally, you need to go to your *'Speed_Hundreds'* sprite where once again you should still have the *'when I receive (Change_LED)'* script that you left behind.

You now need to create another decision making script (shown at the top of the next page) which will set the left ('hundreds') digit on the display to the required costume for each speed setting of the slider.



```

when I receive Update_Speed
  go to x: 215 - Digits_Space * 2 y: -138
  if mBot_Speed = 250 then
    switch costume to Digit 2
  if mBot_Speed = 200 then
    switch costume to Digit 2
  if mBot_Speed = 150 then
    switch costume to Digit 1
  if mBot_Speed = 100 then
    switch costume to Digit 1
  if mBot_Speed = 50 then
    switch costume to Blank
  show
  
```

This is the 'when I receive (Update_Speed)' script required for the 'Speed_Hundreds' sprite.

Note that the last decision in this script (checking if the speed is set to '50') shows the 'switch costume to ()' block set to 'Blank'. This is an additional costume which is required here to show *nothing* if (as in this case) a 'hundreds' digit is not required.

To create this, it is very simple. Just go to the costumes editor and add another costume and name it 'Blank' and then immediately close the costume editor. No painting required, because as its name implies, it contains *nothing*.

Move your speed slider button by clicking just below each speed setting on the backplate and you should see the display change to match the chosen speed.

You already should have tested the speed settings with mBot in step 18; but if you want, you can connect mBot and test them again right now, but first (and as usual at every stage) you should ...

... Save and backup your project once again.

The main component parts of the interface have nearly all been programmed and the only item that you initially installed on your interface screen and which is not yet working is 'mBot_Lamp'. This is the third indicator

lamp which is positioned just to the left of the speed monitor window which shows a Red lamp 'Off' (0) for the first costume or a Green lamp indicating 'On' (1) for the second costume of the sprite graphic.

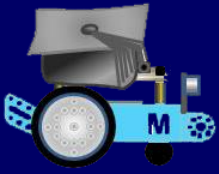
Step 21: - Go to the 'mBot_Lamp' sprite and add the short script shown here on the right. This is another two-way toggle routine of two sprite costumes and is linked to the position of the 'Start/Stop Switch' switch.

You have already added the 'broadcast (mBot_Lamp)' block to each of the four direction arrows and the stop backplate for controlling mBot; so, if you connect mBot and test this now you should see the indicator lamp turn green when mBot is moving and turn red when mBot is inactive - this is not vital, but it does add another nice graphic touch.

```

when I receive mBot_Lamp
  go to x: 125 y: -138
  if Switch2_Pos = 0 then
    switch costume to Lamp_Red
  else
    switch costume to Lamp_Green
  
```

Save and backup your project once more.



Step 22: - As I mentioned at the beginning of this chapter, I decided that adding the day, date and time to my interface might be a nice feature and provide a demonstration of how to add these as graphic output to any project. The 'workings' of the clock rely on the ability of scratch to extract the sub-component parts of the current date (day, month, year) and the current time (hour, minute & second) from the time clock built into all computer systems.

```
define Start_Clock
forever
  if (current hour < 10) then
    set Clock_Hours to join 0 current hour
  else
    set Clock_Hours to current hour
  if (current minute < 10) then
    set Clock_Minutes to join 0 current minute
  else
    set Clock_Minutes to current minute
  if (current second < 10) then
    set Clock_Seconds to join 0 current second
  else
    set Clock_Seconds to current second
  if (current month < 10) then
    set Cal_Month to join 0 current month
  else
    set Cal_Month to current month
  if (current date < 10) then
    set Cal_Day to join 0 current date
  else
    set Cal_Day to current date
  broadcast Update_Clock
```

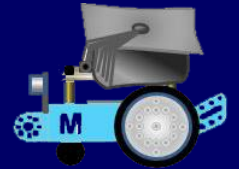
The clock needs a slightly lengthy but not complex script to be created somewhere in the interface project to extract the data from these sub-component parts and store them in variables that can be easily called upon to display suitable output on the interface.

This sort of script would often be triggered by a 'Green-Flag' event but I decided that since the clock on my interface would be activated by clicking the 'Power_On/Off' button then this script should be attached to the power-button sprite.

You need to return to your 'Power_On/Off' sprite which you created and scripted in step 15. Here you created two self-defined 'My Blocks' - 'Operate_Switch' to which you added the scripting it needed to work and 'Start_Clock' which you only defined as a name with no scripting blocks added below the red (automatically created) 'hat' block.

The 'Start_Clock' script needs to work 'forever' so all of the other blocks in this script need to be inside a 'forever' loop 'C' block. Inside it you then need five 'if/then/else' decisions to create the data needed for each of the variables required to store date & time information. The clock always needs to store *two* digits in each variable so each decision checks to see if the value is less than ten and adds a leading zero if it is needed.

The single 'reporter' block `current year` needed for each sub-component; is from the 'Sensing' blocks menu (it's default setting is 'year') - you just choose the name of the sub-component you require from the drop-down menu inside the block.



Once you have completed the first 'if/then/else' decision block (extracting the hour part of the current time) then it is very quick to duplicate it four more times and modify the contents of each to match the script shown on the left of the previous page. The final block that you need to add (inside the 'forever' loop) broadcasts the message to update the clock.

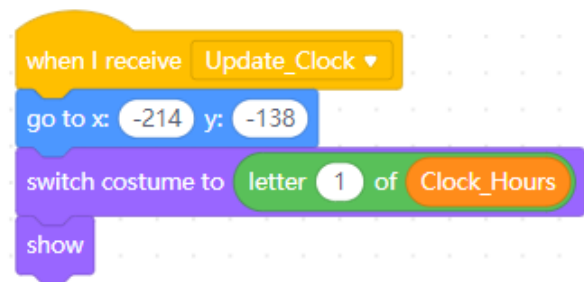
If you want to test that this script works, go to the 'Variables' part of the blocks menu and check the tick-boxes next to the following variables: 'Cal_Day', 'Cal_Month', 'Clock_Hours', 'Clock_Minutes' and 'Clock_Seconds'. These variable monitors will now be visible on the stage. Click the 'hat' block 'Start_Clock' above where you have just added the decision blocks to extract the sub-components of your computer systems time clock. The whole script will 'light up to show that it is running 'forever' and you will see that each of the variable monitors on the stage now contains the data showing the day (date) and month and the current hour minute and second. Click the 'hat' block again and the script will no longer be running. Click the 'hat' block once more and you should see the second and possibly minute variables update themselves. Click repeatedly, and you've got yourself a simple clock; and unless you are doing this at midnight, you will not see the day or year change value!

Save and backup your project file.

Step 23: - Once you have the appropriate variables containing the date and time information that you need; you can get to work on creating digital output to display them in a meaningful way.

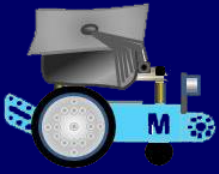
To display the time using 7-segment LED digits in the 'Clock_Monitor' window that you added to your interface is a comparatively easy task. To do this you need six separate digits and two spacers (colons) to display the clock in the format: **00:00:00**. Go to your original 'Sensor_Units' sprite in the sprites pane and duplicate it and rename it as 'Hours_Tens' - this new sprite should contain all of the costumes you added (1 to 9 and 0) and it should also have duplicated the three scripts 'when I receive (Change_LED)', 'when I receive (Update_Digits)' and the self-defined 'My Blocks' 'Digit_Pos' script.

Delete the last two of these scripts but keep the 'when I receive (Change_LED)' script which will enable the new digit to change colour when the colour slider is operated. You then need to create the very short script shown on the right. This positions the digit sprite in the correct position in the monitor window and chooses the correct costume to match the first digit of the current hour.



Since you have duplicated the existing sprite, it will probably be set to the same size (15% in my case) but you will need to set the position of this sprite to x: -214 and y: -138 (and you may also need to move it to the front layer in front of the clock monitor window to enable you to see it). You can now duplicate your new 'Hours_Tens' sprite containing both scripts five more times to make the six sprites required for the clock. Re-name each of the five newly duplicated sprites as:

- 'Hours_Units' positioned at x: -197 and y: -138,
- 'Mins_Tens' positioned at x: -173 and y: -138,
- 'Mins_Units' positioned at x: -156 and y: -138,
- 'Secs_Tens' positioned at x: -132 and y: -138,
- 'Secs_Units' positioned at x: -115 and y: -138



mBot and Me a Beginner's Guide

You should now see all six LED sprites positioned in groups of two inside the clock monitor window. If not, remember to use this block for each sprite in turn:

```
go to front layer
```

```
when I receive Update_Clock
go to x: -197 y: -138
switch costume to letter 2 of Clock_Hours
show
```

The 'Hours_Units' sprite needs its script amending as shown on here on the left. This script positions the sprite in the correct position in the monitor window and chooses the correct costume to match the second digit of the current hour.

```
when I receive Update_Clock
go to x: -173 y: -138
switch costume to letter 1 of Clock_Minutes
show
```

The 'Mins_Tens' sprite needs its script amending as shown on here on the right. This script positions the sprite in the correct position in the monitor window and chooses the correct costume to match the first digit of the current minute.

```
when I receive Update_Clock
go to x: -156 y: -138
switch costume to letter 2 of Clock_Minutes
show
```

The 'Mins_Units' sprite needs its script amending as shown on here on the left. This script positions the sprite in the correct position in the monitor window and chooses the correct costume to match the second digit of the current minute.

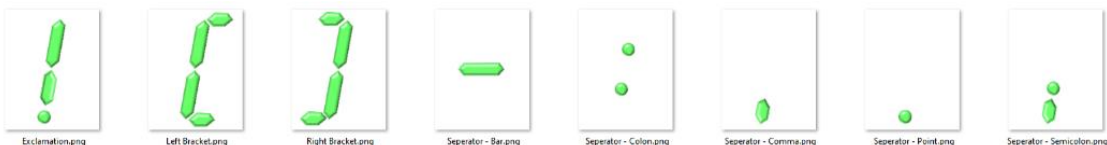
```
when I receive Update_Clock
go to x: -132 y: -138
switch costume to letter 1 of Clock_Seconds
show
```

The 'Secs_Tens' sprite needs its script amending as shown on here on the right. This script positions the sprite in the correct position in the monitor window and chooses the correct costume to match the first digit of the current second.

```
when I receive Update_Clock
go to x: -115 y: -138
switch costume to letter 2 of Clock_Seconds
show
```

The 'Secs_Units' sprite needs its script amending as shown on here on the left. This script positions the sprite in the correct position in the monitor window and chooses the correct costume to match the second digit of the current second.

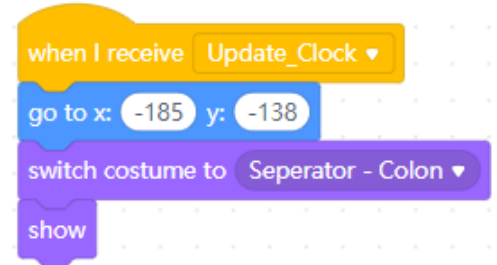
If you click a couple of times on the 'Power_Switch' on your interface, first on and then off; you should see 'working' clock digits appear and disappear accordingly and if you move the colour slider you should also see the colour of the LEDs change. The 'Power_Switch' does stop the clock script too when it is in the 'Off' position. Finally, you need to create two new sprites (colons) to act as separators (:) between hours & minutes and minutes & seconds. You should have created a suitable colon graphic at the same time as you created your numeric digits - the colon should be in the same style too. I made the punctuation set shown here on the right:





To add the two required separators, add a new sprite and add the colon image from your sprites library (or upload it into the library from your .png files first). Name it 'Seperator_1'. Set its size to 15% and its coordinates to x: -185 and y: -138.

The colon separator could just be positioned by setting the above coordinates, but it make sense to adopt the method of always checking its position by adding the short script shown here on the right. You also need to add to this sprite the 'when I receive (Change_LED)' script which will enable the separator to change colour when the colour slider is operated; so go to any of your other LED digit sprites and drag a duplicate of that script across to this sprite.



Next, you need to create the second separator sprite, so just right-click on 'Seperator_1' in the sprites list and click duplicate. The duplicate will automatically be renamed as 'Seperator_2'. All that you have to do here is change the x coordinate in the blue 'go to ()' block in the above script to -144.



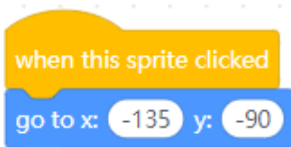
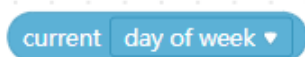
Try clicking the 'Power_Switch' on your interface to turn the clock on - it should be fully working with the hours, minutes and seconds clearly separated.

At this stage, I finally switched my backdrop graphic from 'graph paper' to 'shiny metal'.

Save and backup your project file.

Step 24: - As you can see from the image above, the weekday and the current date are 'engraved' into my metallic backdrop and just like the clock are programmed to change automatically using the variables described in step 22 earlier. The 'engraving' of the data is such a simple trick, it just requires the text to be a slightly lighter shade of grey than the background.

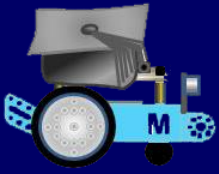
First, add a new sprite and choose 'Paint to open the costume editor and name the sprite 'Weekday'. Set its size to 85% and its coordinates to x: -135 and y: -90. There is no 'reporter' block like those described in step 22 to show weekday names as text but there is one (shown here on the right) which returns the day of the week as a number (1 to 7) with 1 being "Sunday and 7 being "Saturday".



You will need to create two scripts for the 'Weekday' sprite. The first being the usual one-block self-repositioning script (shown here on the left).

The second script will be updated by the broadcast 'Update_Clock' just like the digital clock itself so that the correct day will be chosen automatically by a sequence of seven simple decision processes in the form of "if weekday number is 1 then display costume 'Sunday'" this script is shown on the next page.

As this implies, you will also need seven costumes creating for this sprite using text of a suitable size and colour. Text costumes are a good use of the mBlock5's costume editor (so no need to use Word!).



mBot and Me

a Beginner's Guide

```
when I receive Update_Clock
if current day of week = 1 then
  switch costume to Sunday
  show
if current day of week = 2 then
  switch costume to Monday
  show
if current day of week = 3 then
  switch costume to Tuesday
  show
if current day of week = 4 then
  switch costume to Wednesday
  show
if current day of week = 5 then
  switch costume to Thursday
  show
if current day of week = 6 then
  switch costume to Friday
  show
if current day of week = 7 then
  switch costume to Saturday
  show
```

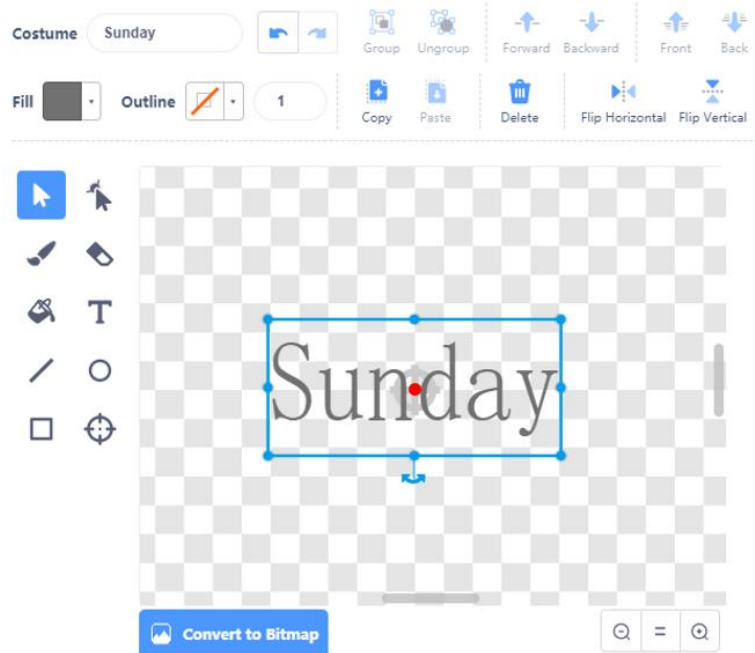
You need to create the seven costumes for this sprite before building the script shown on the left since each 'If' decision needs to refer to a different sprite costume.

In the costume editor ensure that you are in 'Vector' mode not 'Bitmap' mode and then choose the 'T' tool and type the text "Sunday" (without the enclosing speech marks).

Choose 'Serif' as the font and click the 'Fill' box to set the following 'Hue': 0 'Saturation': 0 and 'Brightness': 90. This will make the text a very pale grey. This fill colour suited my backdrop very well, but you might want to adjust the 'Brightness' value to suit your own backdrop.

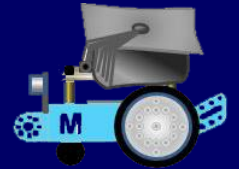
Finally switch to the 'Select' tool (the arrow pointer) and position the text as accurately as you can over the central point of the editor window.

Name the costume that you have made as "Sunday". An image of this shown below:



Duplicate this costume six times in the costume editor and then rename each costume in sequence, one costume for each day of the week. Click on the costume that you have just renamed as "Monday" and edit the text in the middle of the editor screen to say "Monday". You might

want to adjust the brightness of the fill temporarily to check that your text editing is correct. Edit the five remaining costumes in the same way.



Do make sure that you reposition the text of each costume over the central point of the editor window using the 'Select' tool and also reset the fill colour if you changed the brightness. Close the editor and create the script shown on the previous page. This is fairly quick to accomplish; all that you have to do is make one complete 'If' decision and duplicate it six times changing the 'current day of week' number and costume name as appropriate. Click the 'Power_Switch' on your interface to turn the clock etc. on and you should now have the day of the week suitably 'engraved' on the backdrop of your interface.

Making and programming this sprite *is* much quicker than this step suggests by its lengthy description!

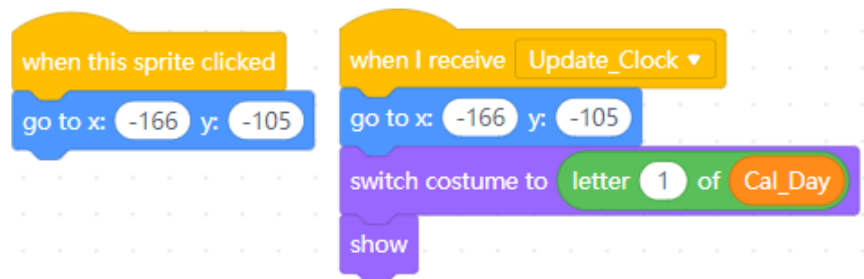
Save and backup your project file.

Step 25: - Adding the date to your interface is very similar in many ways to (and a combination of) the method of adding the weekday or the clock digits that you have just used. The date will be displayed in its simplest conventional numeric form i.e. **dd / mm / yyyy**. As this format would suggest, it needs two sprites to show the two digits needed to display the day (01 to 31), another two digits to display the month (01 to 12) and four digits to display the year. It also needs two separator sprites (/). This makes a rather daunting total of ten sprites in all (25% of the total sprites created in this project) but as usual, these are much quicker to generate than you would think (and note, it *is* possible to complete this step in about 30 minutes).

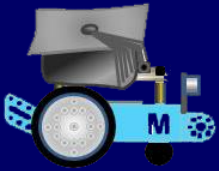
Start by adding a new sprite and then choose 'Paint' to open the costume editor and name the sprite 'Day_1'. Set its size to 80% and its coordinates to x: -166 and y: -105. This sprite needs 10 costumes making for it (1 to 9 & 0) in exactly the same way as you did in step 24 earlier. Name the first costume "1" and then type in the costume editor the text "1" (without the enclosing speech marks). Exactly as you did in step 24, choose 'Serif' as the font and set the fill colour to 'Hue': 0 'Saturation': 0 and 'Brightness': 90. You then need to duplicate this costume nine times more; each costume in turn having exactly the same single digit for both its name and as the text it will display. The last costume will be "0".

You might want to adjust the brightness of the fill for each piece of text temporarily to a darker shade to check that your text editing of each costume is correct.

Close the costume editor and add the two very simple scripts shown here on the right. The first of these is the usual self-repositioning routine. The second is fairly self-explanatory. It chooses the costume (numeric digit) to match the first letter of the data stored in the 'Cal_Day' variable.



It is a fairly simple matter now to duplicate the sprite and name the duplicate 'Day_2' and then change the two blue 'go to x: () y: ()' blocks setting the new x coordinate to -159. You also need to modify the 'switch costume to ()' block to letter 2 of the 'Cal_Day' variable. Click the 'Power_Switch' on your interface to turn the clock etc. on and you should now have the first two days of the current date suitably 'engraved' on the backdrop of your interface below the weekday added in step 24.



mBot and Me a Beginner's Guide

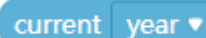
By now you will have a good idea of how all of this works, so my remaining instructions for completing the date display can be much briefer!

Right click on the 'Day_1' sprite in the sprites list and duplicate it twice. Name these duplicates as 'Month_1' and 'Month_2' and edit the scripts of each in turn. 'Month_1' needs the x coordinates both set to -147 and the 'switch costume to ()' block to letter **1** of the 'Cal_Month' variable. 'Month_2' needs the x coordinates both set to -140 and the 'switch costume to ()' block to letter **2** of the 'Cal_Month' variable.

Click the 'Power_Switch' on your interface again and you should now have the two month days of the current date also 'engraved' on the backdrop.

Right click on the 'Day_1' sprite in the sprites list again and this time duplicate it four times. Name these duplicates as 'Year_1', 'Year_2', 'Year_3' and 'Year_4' and as before edit the scripts of each of these in turn. N.B. There is not a variable holding the value of the current year.

'Year_1' needs the x coordinates both set to -127 and the 'switch costume to ()' block to letter **1** of the 'current (year)' reporter block.



'Year_2' needs the x coordinates both set to -120 and the 'switch costume to ()' block to letter **2** of the 'current (year)' reporter block.

'Year_3' needs the x coordinates both set to -113 and the 'switch costume to ()' block to letter **3** of the 'current (year)' reporter block.

'Year_4' needs the x coordinates both set to -106 and the 'switch costume to ()' block to letter **4** of the 'current (year)' reporter block.

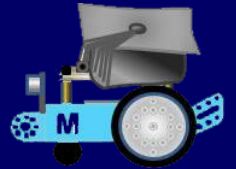
Click the 'Power_Switch' on your interface again and you should now have all of the days of the current date 'engraved' on the backdrop, but they still need the slash separator (/) to complete the effect.

Add yet another new sprite and then choose 'Paint' to open the costume editor and name the sprite 'Separator_1'. Set its size to 80% and its coordinates to x: -151 and y: -105. This sprite needs only one costume made in exactly the same way as the other date sprites. Type in the costume editor the text "/" (the forward slash symbol). Once again, choose 'Serif' as the font and set the fill colour to 'Hue': 0 'Saturation': 0 and 'Brightness': 90. Name the costume as 'Slash' and close the costume editor.

In the scripts pane create another of the usual self-repositioning 'when this sprite clicked' one-block script setting the x coordinate to -151 and the y coordinate to -105. No other scripts are required here.

Duplicate the 'Separator_1' sprite - it should automatically be renamed as 'Separator_2'. Position it at x coordinate -131 and y coordinate -105 and alter the contents of the 'go to' block in the 'when this sprite clicked' script to x: -131 and y: -105. Click the 'Power_Switch' on your interface again and the date should now be separated by the two forward slash symbols.

That's it, your interface is now almost complete - the scripting of its sub-component parts is now complete so save and backup your project file once again.



Step 26: - This is the final step that you need to complete (*and this is a purely cosmetic addition*).

The interface looks better if it has three descriptive labels 'engraved' into it. You actually need to create four new sprites for the three labels required. I found that the power switch label did not show on the backdrop very clearly and there is no way to enhance any text produced using the costume editor; so I decided to use the old trick of creating a 'drop-shadow' effect by positioning a fourth label over the third one with its x & y coordinates offset by 1. The third label being set to a slightly darker shade of grey. To complete this effect, you need therefore two duplicate 'Power_Label' sprites

Using exactly the same techniques as you employed in steps 24 and 25 create a new sprite and then choose 'Paint' to open the costume editor and name the sprite 'Title_Label'. Set its size to 55% and its coordinates to x: 0 and y: 15. This sprite needs only one costume made in exactly the same way as you have done before. Type in the costume editor the text "mBot Speed & Direction Controller" (without the enclosing speech marks). Once again, choose 'Serif' as the font and set the fill colour to 'Hue': 0 'Saturation': 0 and 'Brightness': 90. Name the costume as 'Label' and close the costume editor.

In the scripts pane create another of the usual self-repositioning 'when this sprite clicked' one-block script setting the x coordinate to 0 and the y coordinate to 15. No other scripts are required here. Go into 'Presentation Mode' and you should see that you now have a descriptive title positioned in the centre of the interface between the direction controller sprite and the analogue meter sprite.

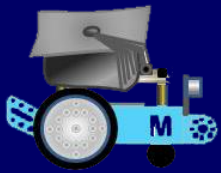
Duplicate your 'Title_Label' sprite and in the costume editor type in the text "ULTRASONIC SENSOR - range in MILLIMETERS". Name the sprite as 'Mm_Label' and set its size to 35% and its coordinates to x: 0 and y: 33. In the scripts pane edit the 'when this sprite clicked' script setting the x coordinate to 0 and the y coordinate to 33. Return briefly to your 'Title_Label' sprite and in the costume editor use the select pointer to *stretch* the text very slightly vertically and then exit the costume editor.

Duplicate your 'Mm_Label' sprite and in the costume editor type in the text "SENSOR - range in CENTIMETERS". Name the sprite as 'Cm_Label' and set its size to 35% and its coordinates to x: 125 and y: 122. In the scripts pane edit the 'when this sprite clicked' script also setting the x coordinate to 125 and the y coordinate to 122. Go into 'Presentation Mode' and check that you can see the sensor labels underneath their respective monitor windows - you will also note that *stretching* the title label has given it a little more emphasis.

Duplicate your 'Cm_Label' sprite and in the costume editor type in the text "POWER ON / OFF". Name the costume as 'Power_Label_1' and close the costume editor. Set its size to 40% and its coordinates to x: -120 and y: 60. In the scripts pane edit the 'when this sprite clicked' script setting the x coordinate to -120 and the y coordinate to 60. Duplicate your 'Power_Label_1' sprite and name the sprite as 'Power_Label_2' and set its coordinates to x: -119 and y: 61. In the scripts pane edit the 'when this sprite clicked' script setting the x coordinate to -119 and the y coordinate to 61. Return to your 'Power_Label_1' sprite and in the costume editor change its 'Brightness' setting to 45. Go into 'Presentation Mode' and check that you can see the two power labels almost, but not quite, on top of each other with a slight hint of shadow on the upper side (N.B. This 'engraving' method produces a much more realistic graphic!).

Well done! - Your interface is complete so save and backup your project file for the last time.

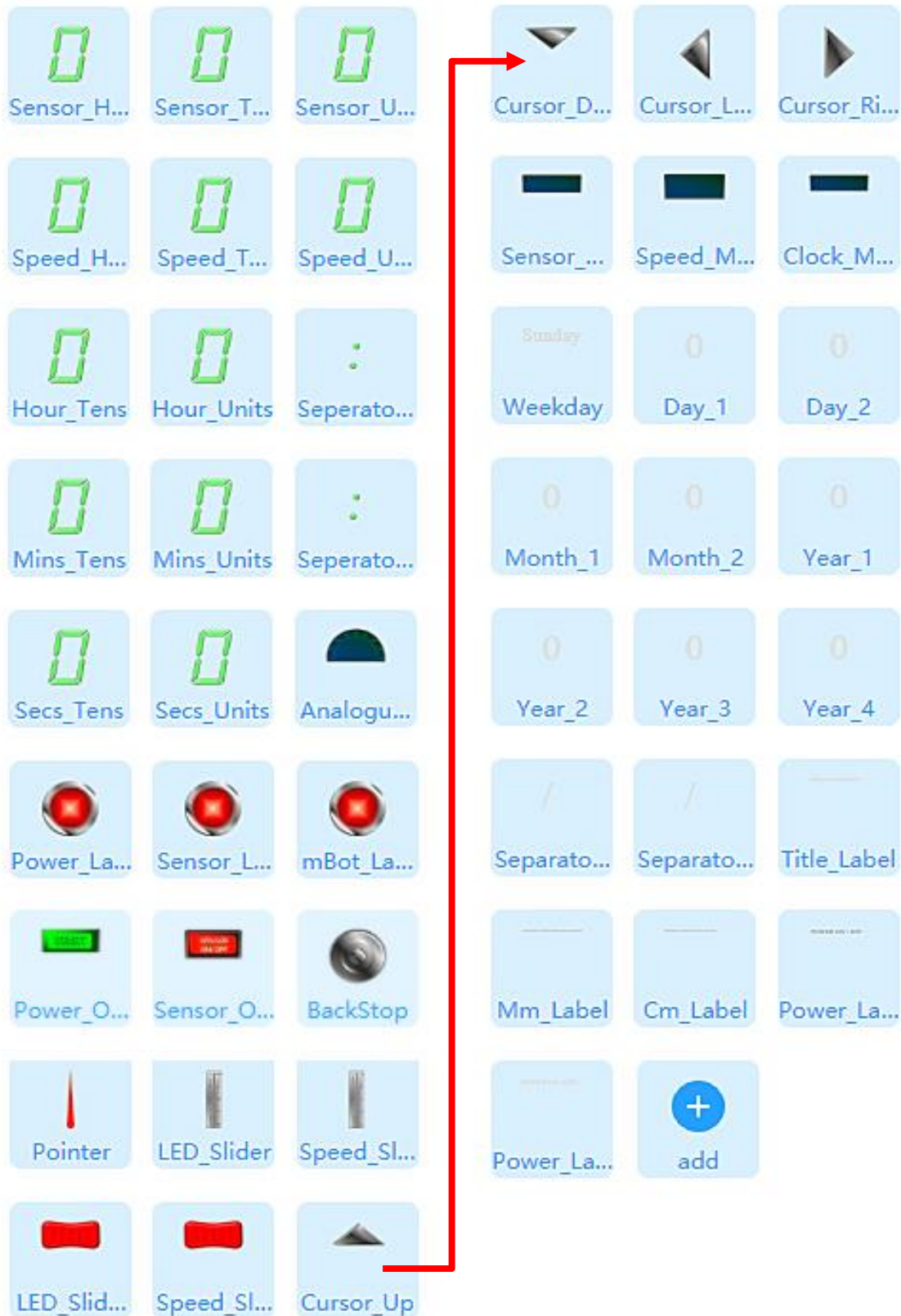
If you can make this project, then you can make anything!

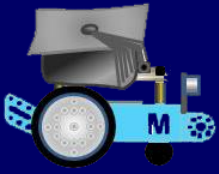


mBot and Me

a Beginner's Guide

Shown below is a screenshot of the complete set of 49 sprites created for my 'Control Interface' project. This is a composite image taken from the sprites list of mBlock 5's edit page. The sprite icons are in the order that I left them at the end of constructing my project.





Chapter 16 - Creating 'Smart Systems' in mBlock 5

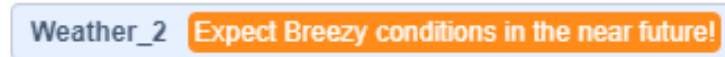
Humans were responsible for creating computers and giving them specific directions in the past, but not necessarily anymore! Computers as machines have always had the power to make decisions according to predefined algorithms built into their programming but new '*smart systems*' techniques enable suitably equipped computer systems to interact with the outside world, execute complicated mathematics very quickly and therefore perform tasks (on their own) that are characteristic of human intelligence.

Artificial Intelligence (A.I.) - the science of mimicking human intelligence using computers is arguably one of the most exciting fields of technological development with far-reaching potential to solve present day problems. Just as humans store information in their brains and use complex mental processes to learn from its patterns; computer scientists have been able to use the processing power of computers to analyze patterns in stored data and allow machines to learn from it very quickly and to remember! This is called '*Machine Learning*' which together with A.I. can be used to build '*Smart Systems*' to interacting with humans by providing suitable feedback based on complex decision-making algorithms.

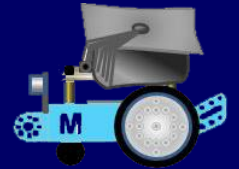
Makeblock are, quite rightly, very keen to expose kids to these computer / real-world interactions and have created several '*Extension*' packs of programming blocks to enable such work to be undertaken using mBlock 5. However, as I mentioned earlier when reviewing these (on pages 50 & 51) the currently available '*Sprites*' tab '*Extension*' blocks are rather limited in what real-world solutions can actually be achieved with them! Nevertheless, experimenting with simple Machine Learning and A.I. feedback using these is still quite good for developing an understanding of some of the basic principles.

Using the '*Video Sensing*' extension together with the '*Teachable Machine*' extension makes it possible to automate Emma's favourite '*Rock, Paper, Scissors*' game - a development that uses a web-cam attached to your computer to sense when a hand movement is made in front of it and then interprets from learned-shapes what hand you have made; the computer then responds by displaying a random hand shape of its own. mBot does not need to be involved, so I built this (as demonstrated later in this chapter) using just '*Scratch Stage Programming*' but I am still not convinced by the accuracy of the machine learning '*recognition*' part.

Using mBlock 5's '*Climate Data*' extension on its own is rather clumsy and seems limited in what it can display easily. The example on the right shows the '*reporter*' block that returns feedback about the weather in 'Oundle, GB'. The single word contained in the feedback from the internet needs to be concatenated (joined) to other words to make a sensible sentence - and then only as a variable displayed in a monitor window on the stage. Such block concatenations can become very lengthy!



By combining such feedback with a complex bit of '*text rendering*' scripting and using the '*Stamp*' block from the very useful '*Pen*' extension to display on the stage and concatenating climate data with linking phrases; and then using the excellent '*Text-to Speech*' extension to read out those phrases almost makes (but not quite) the use of '*Climate Data*' in a '*Smart System*' worthwhile. I am including instructions on how to do this not only because this uses three of mBlock 5's extensions together, but also because it demonstrates some valuable scripting techniques including the aforementioned '*text rendering*' process and a valuable method of '*auto-starting*' a project when it is loaded.



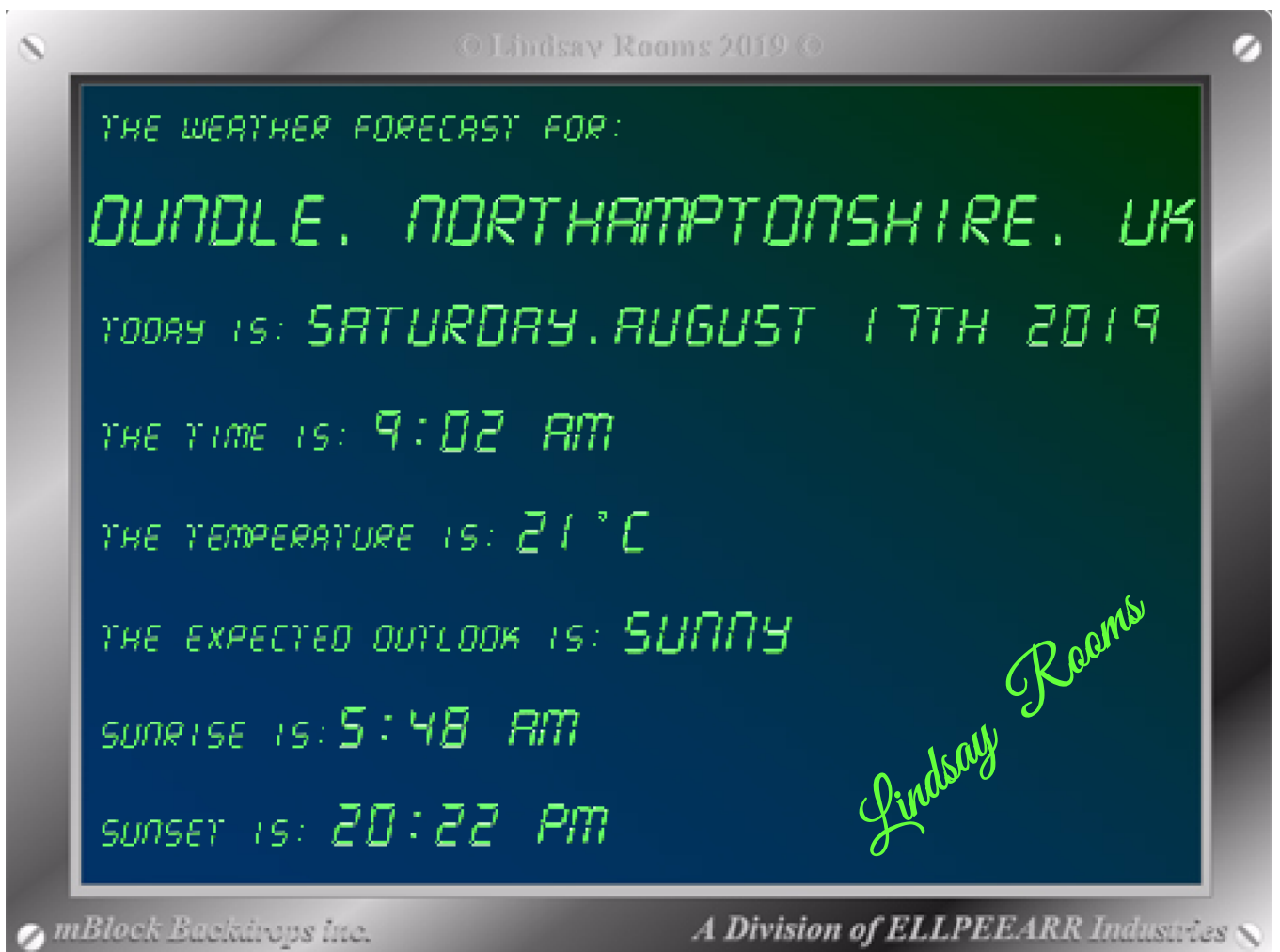
Climate Data with Digital & Spoken Feedback - a 'Smart Systems' development

There *are* some alphanumeric digits available in the sprites libraries, but they are very cartoon like!

To enable this project to work in the way I was envisaging, I had to create yet more .png sprite images to act as costumes alongside the set of numeric 7-segment LED digits that I had already created as part of my preparations for my 'Control Interface' project (see Chapter 15).

I needed a complete alphabet of capital letters rather similar to those shown below:

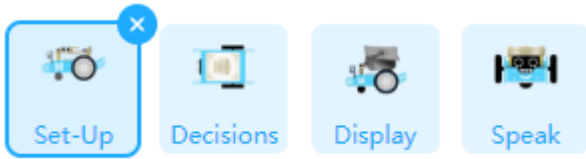
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z



Since I planned on using my *italics* set of numbers, I needed to create an italicized alphabet too. Creating all twenty-six letters from my master no. "8" drawing was not as difficult or as tedious as it might seem since many letters could be created by manipulating the existing segments of my drawing. I also needed to create some punctuation (., ; ! ? & ") made in a matching style - also not difficult! This done, I could begin the scripting required. If you have worked your way through my 'Control Interface' project, then you are certainly capable of this project with the minimum of guidance.



mBot and Me a Beginner's Guide

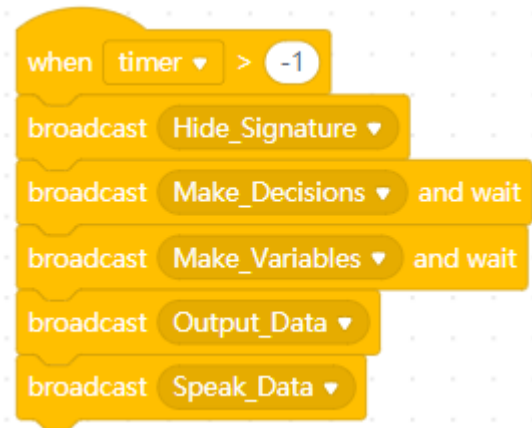


I started by choosing a suitable backdrop and chose the metal-edged 'Monitor Window' that I had drawn in 'Word' earlier.

You could perhaps just create a dark (greeny-black) rectangle in the graphics editor as your own backdrop. I then created the four sprites shown above and named them as 'Set-Up', 'Decisions', 'Display' & 'Speak'.

I used four of my own sprite images for these, but you could use anything else chosen from the sprites library since none of these sprites are actually displayed on the screen. These four sprites are just a convenient way of splitting the many scripts required for the project into sensible groups of blocks that neatly fit into the 'Scripts Area'.

With the 'Set-up' sprite active, I created the following 'start this project' script (in lieu of a 'Green Flag' hat block - see an explanation of this below):



N.B. The 'Timer' is a 'Sensing' feature in Scratch that is set to zero when a project opens and continues to count even when a project stops running. It accurately records in seconds (to three decimal places) how much time has passed.

Clicking the 'Green Flag' or activating the 'Reset Timer' block resets the 'Timer' to zero. It has many programming uses since it cannot be paused, stopped or interrupted.

The '**when (timer) > ()**' hat block provides a cunning way of automatically starting a project if the value in the hat block window is set to (-1). This setting will immediately run any blocks below it (e.g. Broadcasts calling specific scripts elsewhere in the project) and any 'Green Flag' scripts in the project will also auto-start.

The scripts required for this project are all called by using 'Broadcast' messages.

Before creating the auto-running 'start this project' script (shown and described above, I had already created the twenty-one variables that were required to manipulate the climate, the date and the time data required.



The twenty-one variables that I created for this project are shown here on the right:

Many of these variables required populating with data at project start-up, so, on my first sprite 'Set-up' I added the 'Make_Variables' script needed to do this data-populating. This script is shown at the top of the next page.



```

when I receive Make_Variables
set Sunrise_Time to join join Oundle, England, GB sunrise time hour : Oundle, England, GB sunrise time minute
set Sunset_Time to join join Oundle, England, GB sunset time hour : Oundle, England, GB sunset time minute
set Date_Long to join Day join , join join Month join current date join Day_Suffix join current year
set Date_Short to join join current date join : current month join : current year
set Outlook to Oundle, England, GB weather
set Temperature to join Oundle, England, GB highest temperature (°C) *C
set Sunrise_Text to join Sunrise_Time AM
set Sunset_Text to join Sunset_Time PM
set Daylight_Text to join join Sunrise is join Sunrise_Text and join Sunset is Sunset_Text
set Clock_Hours to current hour
set Clock_Time to join join Clock_Hours : Clock_Minutes
set Phonetic_Centigrade to Degrees Centy-grayde
set Phonetic_Temperature to join Oundle, England, GB highest temperature (°C) Phonetic_Centigrade
    
```

As you can see above, many of these variables are using the 'join ()(to ())' block from the 'Operators' category of blocks to concatenate external data with meaningful sentence text.

You will also note that I had to make one block to provide phonetic output; which enabled the 'Text-to Speech' extension to interpret "centigrade" and pronounce it correctly by reading "Centy-grayde"!

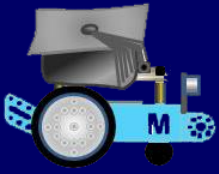
Mostly, the 'Text-to Speech' extension reads text very well, but occasionally you do have to resort to phonetic spelling to make some words sound right. You can test word pronunciation, if you need, to by typing variants of any word into the 'Speak' block and then clicking it.

I then moved on to my 'Decisions' sprite. Here I made the scripts shown on the next page which are all called by the 'Make_Decisions' broadcast. These all use simple 'If ... Then' blocks - each being checked as the script runs until one is true and then updating the appropriate variable.

N.B. Scratch only outputs days of the week and months of the year as numbers so conversion is needed to provide textual output. The main decision to be made here though, is converting the output from the 'Climate Data' weather 'reporter' block; which returns its textual feedback in 'sentence case'.

Since I have not created any 'lower case' character costumes, my project needs the weather report data to be converted into capital letters ('upper case').

Also, (as far as I am aware) there is no definitive list of possible phrases used to describe weather; so I have added to this list of decisions using daily 'trial-and-error'; recording every new phrase generated as the weather report changed (so my list may still need more additions! ?).



mBot and Me a Beginner's Guide

```
when I receive Make_Decisions
  if current day of week = 1 then
    set Day to SUNDAY
  if current day of week = 2 then
    set Day to MONDAY
  if current day of week = 3 then
    set Day to TUESDAY
  if current day of week = 4 then
    set Day to WEDNESDAY
  if current day of week = 5 then
    set Day to THURSDAY
  if current day of week = 6 then
    set Day to FRIDAY
  if current day of week = 7 then
    set Day to SATURDAY

when I receive Make_Decisions
  if current month = 1 then
    set Month to JANUARY
  if current month = 2 then
    set Month to FEBRUARY
  if current month = 3 then
    set Month to MARCH
  if current month = 4 then
    set Month to APRIL
  if current month = 5 then
    set Month to MAY
  if current month = 6 then
    set Month to JUNE
  if current month = 7 then
    set Month to JULY
  if current month = 8 then
    set Month to AUGUST
  if current month = 9 then
    set Month to SEPTEMBER
  if current month = 10 then
    set Month to OCTOBER
  if current month = 11 then
    set Month to NOVEMBER
  if current month = 12 then
    set Month to DECEMBER

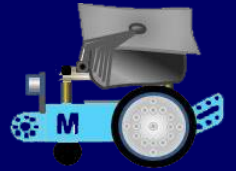
when I receive Make_Decisions
  if current minute < 10 then
    set Clock_Minutes to join 0 current minute
  else
    set Clock_Minutes to current minute
  if current hour > 11 then
    set Clock_Minutes to join Clock_Minutes PM
  else
    set Clock_Minutes to join Clock_Minutes AM

when I receive Make_Decisions
  if Outlook = Sunny then
    set Outlook_CAPITALISED to SUNNY
  if Outlook = Windy then
    set Outlook_CAPITALISED to WINDY
  if Outlook = Breezy then
    set Outlook_CAPITALISED to BREEZY
  if Outlook = Rainy then
    set Outlook_CAPITALISED to RAINY
  if Outlook = Showers then
    set Outlook_CAPITALISED to SHOWERS
  if Outlook = Cloudy then
    set Outlook_CAPITALISED to CLOUDY
  if Outlook = Partly Cloudy then
    set Outlook_CAPITALISED to PARTLY CLOUDY
  if Outlook = Mostly Cloudy then
    set Outlook_CAPITALISED to MOSTLY CLOUDY
```

I decided that I also needed a further, slightly more complex decision making script to add the suffix 'ST', 'ND', 'RD' or 'TH' to the end of the number representing the day in any month.

This was created rather differently by nesting several 'If ... Then' blocks using nested 'Or' blocks inside an 'If ... Then ... Else' block.

This script that I created to do this is shown at the top of the next page.



```

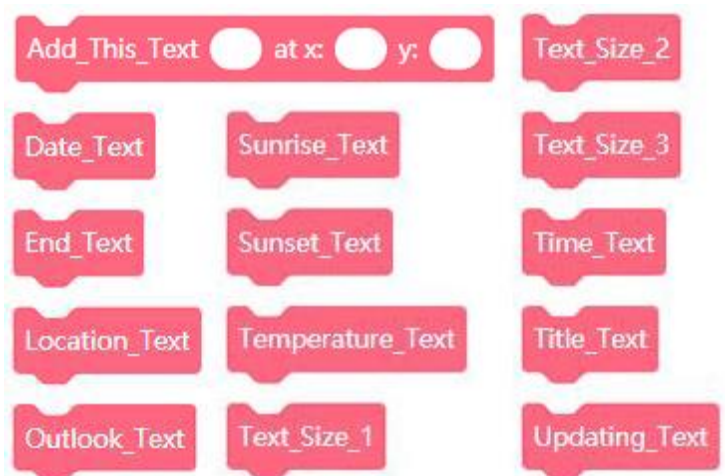
when I receive Make_Decisions
if (current date = 1) or (current date = 21) or (current date = 31) then
  set Day_Suffix to ST
if (current date = 2) or (current date = 22) then
  set Day_Suffix to ND
if (current date = 3) or (current date = 23) then
  set Day_Suffix to RD
else
  set Day_Suffix to TH
  
```

All decisions made; it was time to move on to programme my 'Display' sprite. This is where I created the set of self-defined 'My Blocks' shown below:

The block in the top left-hand corner relies on a clever piece of scripting that I found in the pages of the 'Scratch Wiki' which uses the 'text rendering' script (which is shown on the next page) to 'Stamp' images of alphanumeric costumes at any set position on the stage.

It requires three parameters. The text string to be stamped and the x and y positions where each letter in the string will be stamped.

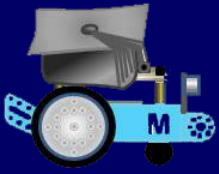
Make this block first because all of the other blocks shown here use it.



For this block to work, the 'Display' sprite where these 'My Block' scripts are being created needs 26 costumes - one for each alphabet letter (from A to Z). Each of these costumes MUST have the same name as the letter it represents.

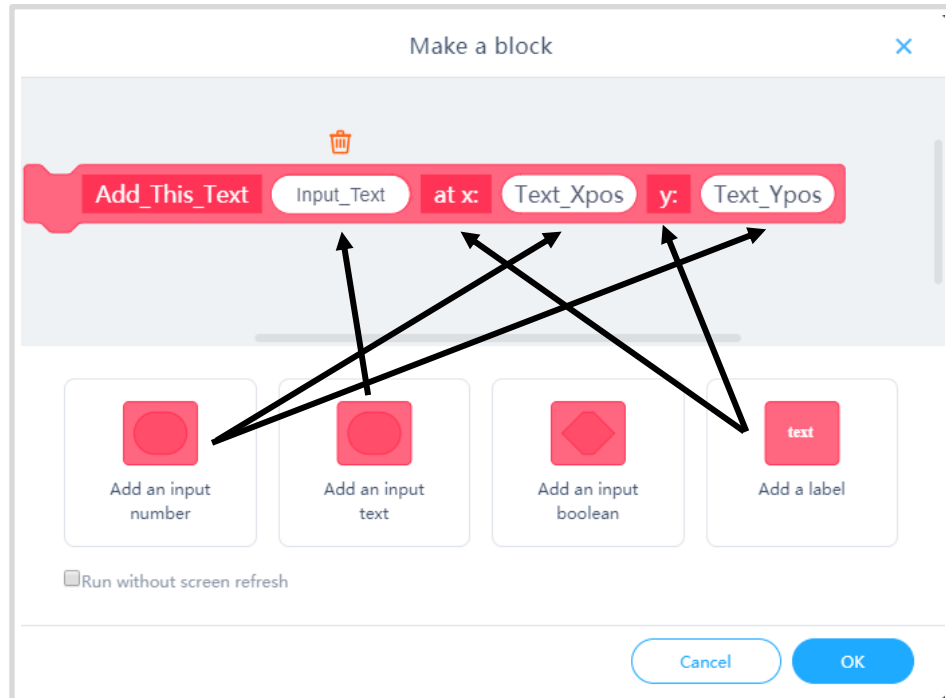
The sprite also needs a further 10 costumes, one for each numeric digit and then a 'space' character (which is just a blank costume!) and its costume name MUST be the equivalent of what you would type to create a space in text i.e. ' ' but note that you can't see it after you have entered it as a costume name!

If other characters need to be created and used (e.g. the punctuation characters that I mentioned) then they also must have a costume whose name is the same as the character.



To make this self-defined block you need to open the 'My Blocks' category and click the 'Make a Block' button. The dialogue box opens with a simple block shape asking for a name for the block - I called it 'Add_This_Text'.

Simple blocks only need a name, but this block needs the five sub-components indicated in the diagram shown on the right - slightly complex, but not too difficult to make.



As you can see from the diagram, I added to the basic block the three required parameters and two helpful labels.

First, I added an 'Input Text' box (to insert the text that needs to be processed by the block) - I called this, predictably, 'Input_Text'. This is, in fact, creating a new variable name inside the block.

I then added a 'Label' into which I typed 'at x:' and then an added an 'Input Number' box which I called 'Text_Xpos' (this is for entering into the block the x coordinate of the starting position of the text to be processed).

```
define Add_This_Text Input_Text at x: Text_Xpos y: Text_Ypos
set Original_Letter_Xpos to x position
set Original_Letter_Ypos to y position
go to x: Text_Xpos y: Text_Ypos
set Stamp_No. to 1
repeat length of Input_Text
  switch costume to letter Stamp_No. of Input_Text
  stamp
  change x by Letter_Spacing
  change Stamp_No. by 1
End_Text
```

I then added a second 'Label' into which I typed 'y:' and then an added another 'Input Number' box which I called 'Text_Ypos' (this is for entering the y coordinate of the starting position of the text to be processed).

Finally, I clicked the OK button to complete the construction of the block.

Note that 'Input_Text', 'Text_Xpos' and 'Text_Ypos' are now available to use in scripts just like named variables.

To this newly self-defined hat block I added the other blocks needed to complete the 'text rendering' and 'Stamping' script which is shown here on the left in its entirety.

This script sets the starting point of the text to be stamped and sets the stamping position to 1.



It then calculates the length of the text, chooses the letter (costume) to be stamped and moves the next stamping position by a predefined 'space'; finally increasing the stamping number by 1.

The stamping routine is repeated until the total number of text characters in the string is reached. The last block in this script is the self-defined block which I called 'End_Text' (this script is shown on the right). The 'go to x:() y:()' block alone could have been at the bottom of the previous script, but I wanted to reset the sprite

costume back to my default 'Teacher_bot' costume (instead of the last used costume showing). A neat but important touch - well I thought so anyway! I next created three more self defined block scripts to set the size of any text to be rendered. These simple scripts are shown below:

```

define End_Text
go to x: Original_Letter_Xpos y: Original_Letter_Ypos
switch costume to Teacher_bot
go to x: 150 y: -110
set size to 50 %
hide
    
```

```

define Text_Size_1
set Letter_Spacing to 8
set size to 5 %
    
```

```

define Text_Size_2
set Letter_Spacing to 13
set size to 8 %
    
```

```

define Text_Size_3
set Letter_Spacing to 15
set size to 10 %
    
```

I then created the blocks that produce all of the individual lines of text to be rendered using the clever 'Add_This_Text' block script shown and described on the previous page:

```

define Title_Text
Text_Size_1
Add_This_Text THE WEATHER FORECAST FOR: at x: -200 y: 135

define Location_Text
Text_Size_3
Add_This_Text OUNDLE, NORTHAMPTONSHIRE, UK at x: -200 y: 100

define Date_Text
Text_Size_1
Add_This_Text TODAY IS: at x: -200 y: 58
Text_Size_2
Add_This_Text Date_Long at x: -120 y: 60

define Time_Text
Text_Size_1
Add_This_Text THE TIME IS: at x: -200 y: 18
Text_Size_2
Add_This_Text Clock_Time at x: -95 y: 20

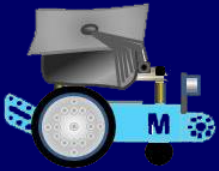
define Temperature_Text
Text_Size_1
Add_This_Text THE TEMPERATURE IS: at x: -200 y: -20
Text_Size_2
Add_This_Text Temperature at x: -40 y: -18

define Outlook_Text
Text_Size_1
Add_This_Text THE EXPECTED OUTLOOK IS: at x: -200 y: -58
Text_Size_2
Add_This_Text Outlook_CAPITALISED at x: 0 y: -56

define Sunrise_Text
Text_Size_1
Add_This_Text SUNRISE IS: at x: -200 y: -95
Text_Size_2
Add_This_Text Sunrise_Text at x: -108 y: -92

define Sunset_Text
Text_Size_1
Add_This_Text SUNSET IS: at x: -200 y: -132
Text_Size_2
Add_This_Text Sunset_Text at x: -110 y: -129
    
```

Switching to the 'Speak' sprite it is at last time to make the project talk. Use the 'Text-to Speech' extension to create a 'Speak_Data' script like the one shown at the top of the next page.



mBot and Me a Beginner's Guide

```

when I receive Speak_Data ▾
  speak The Weather Forecast For
  speak Oundle
  speak Northamptonshire, UK
  speak join Today is Date_Long
  speak join The Time is Clock_Time
  speak join The Temperature is Phonetic_Temperature
  speak join The Expected Outlook is Outlook
  speak Daylight_Text
  broadcast Show Signature ▾

```

Since the bottom of my stage display screen looked a little empty next to the short lines showing the Sunrise and Sunset times, I decided to add a new (fifth) sprite which I called 'Signature' to 'sign' the bottom corner of the screen.

The last block in the 'Speak_Data' script (shown here on the left) calls the 'Show_Signature' script sprite (see below).

I added a scan of my signature as a costume for the sprite and then added the two scripts shown below to it.

N.B. For security purposes, I have replaced my signature in the graphic of my project screen shown on page 138 with a rather poor text version which does not really do full justice to the original!

You can completely omit this signature sprite if you want to.

Finally, I added the 'Updating_Text' block and the two hat block scripts shown below.

The 'Output_Data' broadcast received block clears the screen and then triggers all of the self-defined text rendering blocks to run in the sequence that you can see on the left of the two scripts shown below.

```

when I receive Show_Signature ▾
  show
  set brightness ▾ effect to 20

```

```

when I receive Hide_Signature ▾
  hide
  erase all
  broadcast Show_Updating ▾

```

```

when I receive Output_Data ▾
  erase all
  Title_Text
  Location_Text
  Date_Text
  Time_Text
  Temperature_Text
  Outlook_Text
  Sunrise_Text
  Sunset_Text

```

```

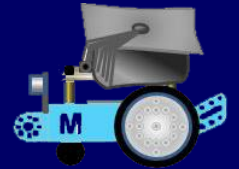
when I receive Show_Updating ▾
  Updating_Text
  define Updating_Text
  Text_Size_3
  Add_This_Text UPDATING... at x: -70 y: 0

```

N.B. The text 'Updating ...' appears in the middle of the stage display only whilst the climate data is being processed.

The 'Show_Updating' broadcast is, as you can see above, is triggered by the 'Hide-Signature' broadcast.

You may need to rethink how to call this script if you omit the signature sprite.



You versus the computer! - a 'Smart Systems' development of 'Rock, Paper, Scissors'

Emma used to enjoy the simple game of 'Rock, Paper, Scissors' and as you have seen earlier, we used the game concept to develop several computerized variants with, or without, mBot's input. In this version, which uses a web-cam attached to your computer, the camera senses when a hand movement is made in front of it and then responds by displaying a randomly chosen hand-shape graphic of its own.

mBot itself does not need to be involved here, but in a continuation of the earlier game projects I have named "mBot" as the second player (although it *is* the computer making all of the decisions).

Outcome

Player_Number

Random_Choice

Random_Number

Response

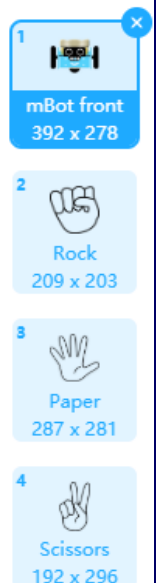
Once again, there is no 'Green Flag' script to start this project and once again it relies on broadcasts to trigger events. You need to begin by creating the five variables shown here on the left.

Next, you need to create the four sprites shown on the right. The first one I called mBOT and this is where the scripts which trigger 'smart' decision making all take place.



Into this sprite I added the three costume graphics used in my last 'Rock, Paper, Scissors' project. (see Chapter 12 - page 74). These costumes are shown here on the right. I decided that the look of this project should remain very simple and retain the 'drawn on a whiteboard with a felt-tip pen' look for these costumes. So, my backdrop is blank - just plain white - and you can (if you want to) choose your own 'look' for this project.

The other three sprites shown above (which I named 'Player', 'Computer' and 'Outcome') are there just to separate out the scripts that make specific game decisions. They all have no graphic showing because the first costume in each sprite is a 'blank' costume.

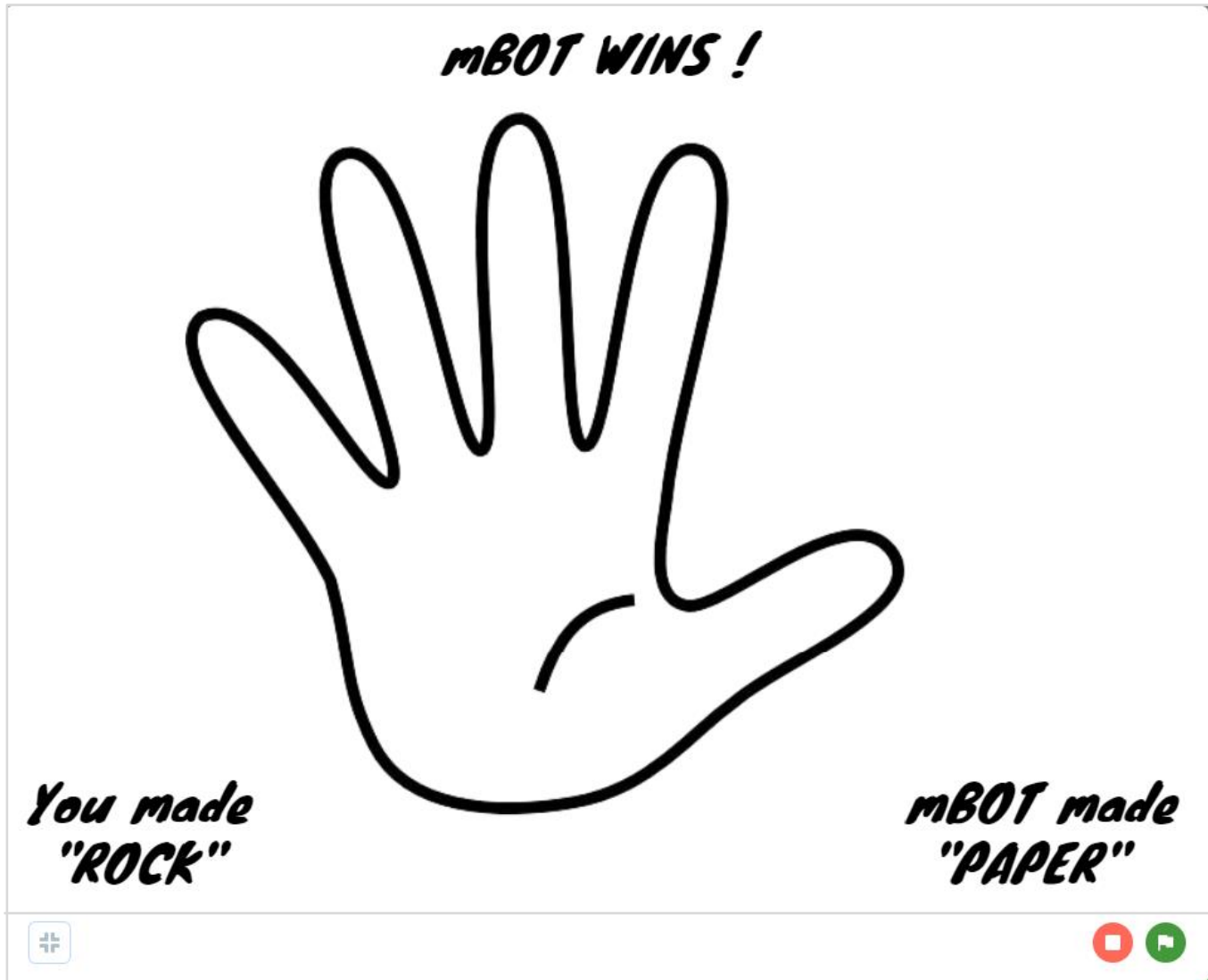


Each of these sprites has (more-or-less) the same four costumes; all created in the graphics editor. The first being a blank costume to enable the clearing of data from the screen. The remaining three 'Player' costumes have text written in 'Marker' font in the centre of the display. The first of these saying, 'You made **ROCK**' is shown here on the left.

It was easy to copy this costume twice more and alter the 'You made nnn' text to '**PAPER**' and '**SCISSORS**' in each; and then so simple too to add these to the 'Computer' sprite where each occurrence of 'You made' was replaced with '**mBOT made**' (- this could have been 'the computer made' but using 'mBot' seemed best). The same costumes were then copied to the 'Outcome' sprite and the text in each changed again to '**PLAYER Wins!**' '**mBOT Wins!**' and '**It's a DRAW!**'. It is also just as easy to create the single decision-making script needed for each of these three sprites once and then duplicate & modify it for the other two sprites.



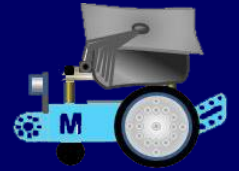
My 'drawn on a whiteboard with a felt-tip pen' project screen layout (with the computer's random selection of a hand-movement graphic together with the feedback text from the three text output sprites) is shown below:



No programming was required for positioning the three text feedback sprites or the hand-movement graphics - I just went to each sprite in turn and dragged each about on the stage (by trial and error) until they looked as intended - simple, but effective.

At the top of the next page are the three scripts that select the costume for each of the text feedback sprites ('Player', 'Computer' and 'Outcome'). The 'Player' decision is made by the response from the web-cam; the 'Computer' decision is made by random-number generation and the 'Outcome' decision is made by a calculation decision based on the other two.

Make the first script (on the 'Player' sprite) and then duplicate it across into the other two sprites. Change the 'reporter' variable that is being compared in each 'If' decision as shown in the scripts below, first modifying the 'Computer' script and then the 'Outcome' script.



These three sprites scripts are all activated by the 'Update_Feedback' broadcast which will then display the three sets of 'chosen' text costumes on the stage.

Player Sprite Script

```

when I receive Update_Feedback
if Player_Number = 0 then
switch costume to 0
if Player_Number = 1 then
switch costume to 1
if Player_Number = 2 then
switch costume to 2
if Player_Number = 3 then
switch costume to 3
    
```

Computer Sprite Script

```

when I receive Update_Feedback
if Random_Number = 0 then
switch costume to 0
if Random_Number = 1 then
switch costume to 1
if Random_Number = 2 then
switch costume to 2
if Random_Number = 3 then
switch costume to 3
    
```

Outcome Sprite Script

```

when I receive Update_Feedback
if Outcome = 0 then
switch costume to 0
if Outcome = 1 then
switch costume to 1
if Outcome = 2 then
switch costume to 2
if Outcome = 3 then
switch costume to 3
    
```

Check_mBOT

Check_Player

Check_Random

Check_Video

Compare_Results

Reset_Icon

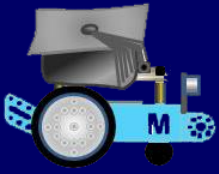
Now it's time to create the complex scripting required for the 'mBOT' sprite. First, you need to create the six self-defined blocks shown on the left. On the right is shown the primary script that kicks everything off, so you need to make this next.

There is no 'Green Flag' script to start this project (and the auto-run method described earlier is not used either). Instead, the 'when video motion > ()' hat block auto-runs the project; waiting for any movement to be detected by the web-cam and if movement is detected then the script continues to run; if no movement is detected then it just waits.

Putting 10 in the block (its default setting) makes it quite sensitive to any movement - you may want to experiment with higher numbers to lower the detection threshold. Setting the video to 'on' is sensible and setting 'video transparency' to 100 hides webcam output completely. (N.B. a setting of 0 displays what your webcam can see on the stage!). Before proceeding any further, you need to train your web-cam to recognize your own hand movements.

```

when video motion > 10
turn video on
set video transparency to 100
Check_Video
Check_Random
Compare_Results
show
broadcast Update_Feedback
wait 4 seconds
hide
Reset_Icon
set Player_Number to 0
set Random_Number to 0
set Outcome to 0
broadcast Update_Feedback
    
```



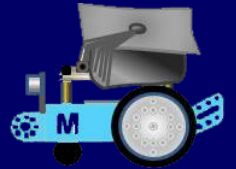
The principle of what is happening here is right, but I'm still not convinced how good this really is; nevertheless, it seems to work - sometimes!

The TM category only has one block which accesses a *'Training Model'* window. This also opens an integral on-screen *'Recognition Window'* showing what your web-cam can see.

You need to name three *'categories'* (rather like variables) which, when the model is in use, will become available as new blocks added to the extension. These *'categories'* are *'trained'* on what the web-cam can see. Name each category *'Rock'*, *'Paper'* and *'Scissors'* and then click the *'Learn'* button for the first category and a little thumbnail of the web-cam image will be shown.

Make the hand-shape several times and the *'Recognition Window'* should start to identify the shape with the category name. Repeat for each of the categories several times until the *'Recognition Window'* seems to identify each shape with a good percentage of certainty. Click the *'Use the Model'* button to exit training mode.

You now have access to the *'boolean'* block *'recognition result is ()'* which has a drop down list of your three named categories. You can now add these blocks to the self-defined block *'Check_Video'* (shown on the next page) which sets a numeric value for each hand-movement identified by the web-cam.



Finally, to complete the project, you need to create all five of the self-defined block decision-making scripts; and the short script (to reset the 'mBOT' sprites default icon) which are all shown on the next page.

These are the five self-defined block (decision making) scripts and the reset default icon script.

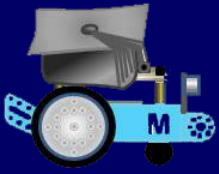
```

define Check_Video
if recognition result is ROCK ? then
set Player_Number to 1
if recognition result is PAPER ? then
set Player_Number to 2
if recognition result is SCISSORS ? then
set Player_Number to 3

define Compare_Results
if Player_Number = Random_Number then
set Outcome to 3
Check_Player
Check_mBOT

define Check_Random
set Random_Number to pick random 1 to 3
if Random_Number = 1 then
switch costume to Rock
set Random_Choice to ROCK
if Random_Number = 2 then
switch costume to Paper
set Random_Choice to PAPER
if Random_Number = 3 then
switch costume to Scissors
set Random_Choice to SCISSORS

define Reset_Icon
switch costume to mBot front
hide
    
```

mBot and Me

a Beginner's Guide

define Check_Player

if $\text{Player_Number} = 2$ and $\text{Random_Number} = 1$ then

set Outcome ▾ to 1

if $\text{Player_Number} = 2$ and $\text{Random_Number} = 3$ then

set Outcome ▾ to 1

if $\text{Player_Number} = 3$ and $\text{Random_Number} = 1$ then

set Outcome ▾ to 1

if $\text{Player_Number} = 3$ and $\text{Random_Number} = 2$ then

set Outcome ▾ to 1

define Check_mBOT

if $\text{Player_Number} = 1$ and $\text{Random_Number} = 2$ then

set Outcome ▾ to 2

if $\text{Player_Number} = 2$ and $\text{Random_Number} = 3$ then

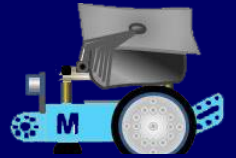
set Outcome ▾ to 2

if $\text{Player_Number} = 3$ and $\text{Random_Number} = 1$ then

set Outcome ▾ to 2

if $\text{Player_Number} = 2$ and $\text{Random_Number} = 3$ then

set Outcome ▾ to 2



Chapter 17 - About Good Habits & Best Practice

I have so far in this book been trying to demonstrate how I have found my own way through the maze of information (or lack of it) about mBot and more particularly, mBlock 5. I feel that the time has come to preach what I have tried to practice so far. Any worthwhile project needs you to develop quite a few skills and habits as you progress on your own journey.

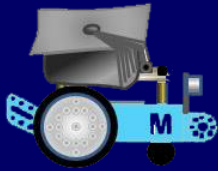
- Always aim for small easy to understand procedures preferably called by broadcast messages - keep 'Green-Flag' activation to a minimum or non-existent.
- Always use a sensible and consistent naming convention for projects, scripts, broadcasts and self-defined blocks. When you assign names, try to make sure that they are accurately descriptive and logical. They *MUST* be understandable to either someone else or indeed yourself when you return to a programme that you may have scripted months or even years earlier. I always use the underscore character “_” as a separator to represent a space (making one continuous set of characters) e.g. *using_an_underscore_to_simulate_a_space*. This prevents errors if moving your work across into textual (*Python* or *Arduino*) programming.
- Once you have an idea for a project, do carry out some systems design and planning (inc. initial sketches and logical sequencing algorithms).
- Develop a variety of graphics programming techniques and then always use high-quality graphics to enable you to create and add to your own bespoke sprites toolkit.
- Always try to test-bed individual sub-component parts of a project.
- You *MUST* develop good housekeeping techniques too. Do aim to keep on top of all your files inc. their storage to hard disk (always using the grandfather, father, son principle for saving backup copies). Sensible storage of graphics files, sprites files and the valuable .sprite3 files are vital too.

A very good habit to develop when writing scripts is to use 'Comments' call-out boxes frequently to annotate your projects and make them understandable. It is important to note that descriptive 'comments' can be usefully added to the mBlock 5 'Scripts Area' without them being connected by leader lines to individual blocks.

I do use comments in my projects a lot, but I have omitted them from most of the script sequences demonstrated here since they are fully described in each project sequence.

If you add a 'comment' to an individual block, then the leader lines are attached to the right-hand end of most blocks (but from the middle of a 'My Blocks' hat block). If blocks are nested inside other blocks, then the leader line is attached to the right-hand end of the nested block which was right-clicked to add the comment.

'Comments' call-outs now look much neater, sharper and clearer than they ever were in mBlock 3. The leader-lines now remain connected to the block that the comment refers to too and they can now be positioned *anywhere* with the leader lines at *any* desired angle (not just horizontal like before!) - much improved, but not yet perfect! Sadly, they still tend to move and reposition themselves, particularly if you open and then close the code comparison windows which pull in or out from to the right edge of the interface.



About mBlock Files

mBlock 5 filenames in your 'My Projects' window (in the Makeblock Cloud) show a maximum of fifteen characters (but only fourteen when you are editing them). When developing a project, I always save my files with short filenames (10 chars. max.) appending v. 1, v.2 etc. to each modification of a project. In this way the whole filename can be seen under its thumbnail including which version it is.

I have found this useful - even though the file thumbnail also shows the info. (minutes days or weeks since last saved).

'My Projects' is just one large storage area and any files stored there will are always visible with no way to logically subdivide or order them. You have to sign-up or sign-in to mBlock to use their cloud service. To do this, you click the *sign up / sign in* icon at right-hand end of the tool bar.

If you double-click (or right click) on a mBlock 5 filename in the cloud and choose to edit the name, then it becomes highlighted, showing that it has been selected. Unlike other forms of text editing you cannot click again to position an 'insert point' to edit just part of the filename. This is a little tedious, but there is a way round this. Just press one of the cursor keys on your keyboard. This will deselect the highlighting of the filename slot but remain in editing mode and you can now move the 'insert point' to anywhere in the filename to edit it. 'Up' moves the cursor to the start of the filename, 'Down' moves it to the end of the filename; whilst 'Left' and 'Right' move the insert point one character in the indicated direction.

N.B. Long text strings in 'input slots' / 'bubbles' in mBlock 5 programming blocks obey the same editing rules described above.

mBlock 5 does not use the standard Windows format (three-letter, occasionally four letter) extension for files - it uses *six* letters for its Windows PC file extension (*.mblock*) and it only recognises this as well as the older *.sb2* and *.sb3* file extensions and the *.json* - JSON file format (JavaScript Object Notation) which is a standard data interchange format primarily used for transmitting data between a web application and a server.

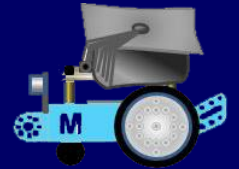
Do be aware that if you try to open an old *.sb2* file (created in mBlock 3) in mBlock 5 then the file *may* open, but without any of the *robotics* blocks originally in that file showing in the 'Scripts' area.

mBlock 5 now (since the July v5.1.0 update) also recognises the very useful (but unusual) *.sprite3* extension. This is used to save sprites, and rather more importantly their attached block scripts, costumes and comments.

If you have installed the mBlock 5 application to its default location on a PC, then it's hard to locate the mBlock 5.exe file if you want to choose the App required to open mBlock 5 files using the 'always open with this file type' dialogue.

It is well hidden, but I looked at the desktop shortcut icon's properties to see the path to the .exe file and found it hidden in an 'invisible' folder on the C drive of my PC. The full default path to the .exe file is as follows:

C:\ProgramData\mBlock5\mblock\mBlock.exe



To see it, you need to choose 'View' from the menu at the top of the primary C drive folder and then check the box, 'Hidden items'. You could perhaps install the software to a more accessible and visible folder instead?

If you use mBlock 5's 'Save to your computer' option, files will be automatically added to the last folder you used in your own filing system - unless you reset the file path to your folder of choice. Project files are otherwise saved to 'My Projects' in the mBlock cloud for quick access.

The Makeblock Cloud is nearly always fast and reliable, but it does 'hang-up' & stop sometimes. I would always recommend using the 'Save to your computer' option as a secondary storage method and I do use the 'My Projects' cloud storage option with my own mBlock 5 files - BUT - I always make a disk backup frequently too. It makes sense to only use 'My Projects' for work in progress; and then emptying out by archiving your files to storage in folders on your hard-disk when completed. As long as you sign in to mBlock 5, either on your PC, using a web browser, or the mBlock mobile app. you can always find your ongoing projects in your 'My Projects' page in the Makeblock cloud from anywhere!

If you, like I did, started your programming journey using mBlock 3 then any files you created there are redundant and need to be totally reworked in mBlock 5 should you still need to use them. There is no conversion process available and it seems that old **.sb2** files *can* only be opened in mBlock 5 by changing the extension to **.mblock** - but it's hardly worth the trouble since they often fail completely or do not transfer correctly. The main problem with moving files across from one system to the other seems to be with the fact that mBlock 5 now has two separate programming areas (the 'Devices' tab and the 'Sprites' tab) whereas mBlock 3 just had one programming area for both robotics block programming and sprite programming.

Early on in my experiments with the new application I found that it was not worth trying to open any of my old files this way in mBlock 5 but I did find however, that I could open mBlock 3 and copy text from any comments boxes to a temporary text file and then paste the text into new comment boxes attached to blocks in mBlock 5 - this saved time, but was only worth it if they contained a lot of detail!

So on-the-whole, totally rewriting scripts in mBlock 5 is actually quite quick, especially if you capture a screenshot (in mBlock 3) of any block scripts that you want to recreate and then rebuilding them in mBlock 5 using the screenshot (or a print-out) as a guide.

Creating a 'Set-Up' File

A useful way of starting mBlock 5 is to make mBot your 'Regularly Used Device', making it the default device (not 'Codey') set to appear each time you open a 'New' project. Open the 'Device Library' list and hover the cursor over the top left corner of the mBot device icon. A hollow blue star will become apparent, click the star to make it turn solid blue - this will make mBot the default start-up device.

It is even quicker to get going with mBlock 5 by setting up the way you like to use it and then saving this set-up as a file on your computer for re-use, adding a shortcut to this file on your desktop. Every time that I use the Block 5 application, I now choose a shortcut icon on my desktop to open a file saved as '*mBot Setup Page*'.

I also have a copy of this file saved in 'My Projects' in the cloud.



To create this file, (if you haven't yet used the 'Regularly Used Device' adjustment described above), open mBlock 5 and from the 'Devices' tab, open the device library and add the mBot sprite as a new device. Then deleted the 'Codey Rocket' sprite as an available device leaving mBot as the only icon in the 'Device' sub-panel. Switch to the 'Sprites' tab and delete the default 'Panda' sprite, replacing it with a sprite of your own choosing. I used one of my own mBot drawings - not crucial, but it looks good.

Change the name of this new 'Sprite' to 'Set-Up' both in the editor and in the 'Sprites' tab and finally add a suitable background image to give the stage some meaning - always better than it remaining white! I used my 'Graph Paper' backdrop. It's also a good idea to add the useful 'Makers Platform' extension. Save the file (I called mine 'mBot Setup Page') and make a desktop shortcut. See Chapter 5 (page 21) and Chapter 9 (page 42) for more on my setup page.

Setting up a Sensible Filing System

It may seem presumptuous for me to suggest this but having a sensibly organised file hierarchy is a very good habit to adopt. When I started my mBot programming journey I set up an 'mBot Stuff' folder in my own user documents in my PC's filing system. I added the folders shown below to organise my files:





The 'Scripts (Sprite 3) Files' folder contains my saved **.sprite3** files.

These files are VERY useful because they are a way of storing and retrieving scripts for future use but by default they have the plain 'unknown file type' icon (shown on the right). Since these are not immediately linked to mBlock5 I had to right-click one of them and choose the app that would always open the **.sprite3** file extension. It's not crucial to do this since you import these files into mBlock 5 when needed, but I did it anyway and the plain icon for all of these was replaced by mBlock's 'Panda Cog' icon.



Dancing Cat Scripts.sprite2

The sub-folders within my 'Project Files' folder are also shown as part of the filing system diagram on the previous page - some of these folders have sub-folders too.

My 'Sprite Files' folder is quite important as it stores (in several sensibly named sub-folders) the sprites making up my *oh-so-useful* sprites toolkit.

My 'Sound Files' folder stores a few sounds - but not many are needed for robotics. Sound files do have their uses but do remember that they play through your computer's speakers and not through mBot. If you click on the 'Sounds' tab in mBlock 5 you will see that there is just one default sound file, 'Pop'. I deleted this from my 'mBot Setup Page' file too before saving it. It is easy to add more sound files as, when and if, you really need them in a project.

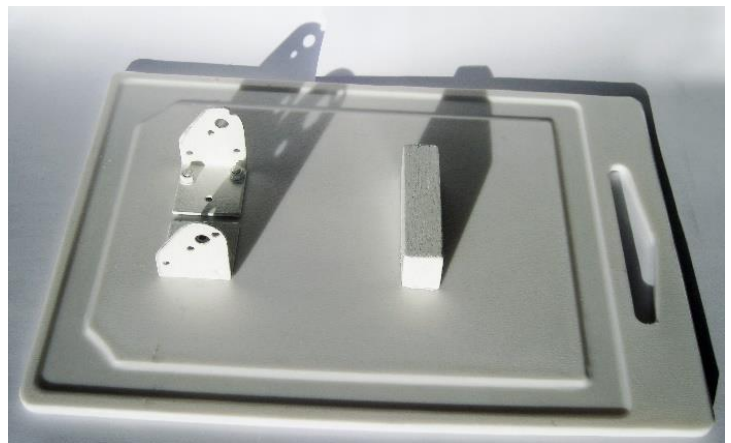
Making a purpose-built Work Tray

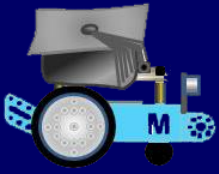
Raising the back of the robot up off a desk or table-top is easy to do if you put something about 20mm high under the area (where the back axle would be if only it had one!). This use of an 'axle-stand' is highly desirable and avoids mBot thrashing about if it does go amok and for some reason you unintentionally start-up its drive motors.

It's not too difficult however to make a special worktop tray, a 'service bay' (or 'sick-bay' perhaps) to mount the robot with its wheels raised. A board with a small block of wood screwed on to it can provide a permanent 'axle-stand' and if the front of the robot is temporarily attached to the board by a couple of bolts put through the big (8mm dia.) holes it will hold and steady mBot; which is ideal when modifying & testing new components.

I made a purpose-designed 'service-bay' tray which I use for most programme testing until I know the robot runs as it should. (see the picture on the right & the picture of it in use on the next page).

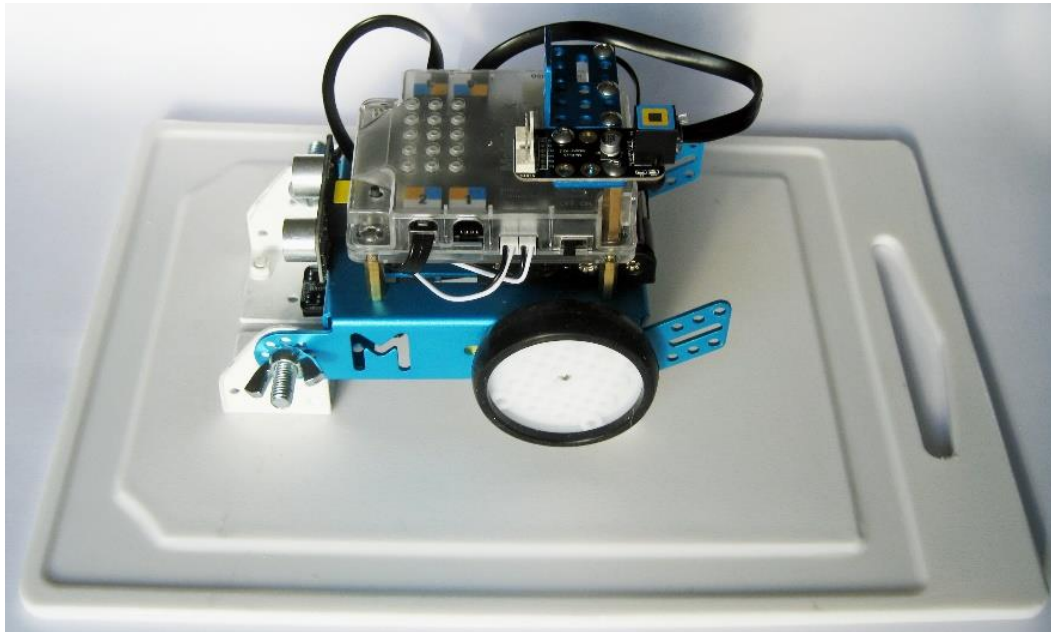
It uses an 'Apollo' brand rectangular 35 x 25cm Polypropylene Chopping Board – approx. £5 from TESCO & £6.50 via Amazon; and a couple of filed-to-suit angle brackets (about £4.50 from B&Q).





mBot and Me *a Beginner's Guide*

The grooves around the board retain screws very usefully when assembling extra components.



I also bought some extra M4 nuts, some M4 x 12mm (longer than the original 8mm) screws and some M4 wing nuts to help with constructing robot models - the cost was about £5.50 in total for 20 of each on eBay.

I have quite big fingers and M4 nuts are quite small, so when model

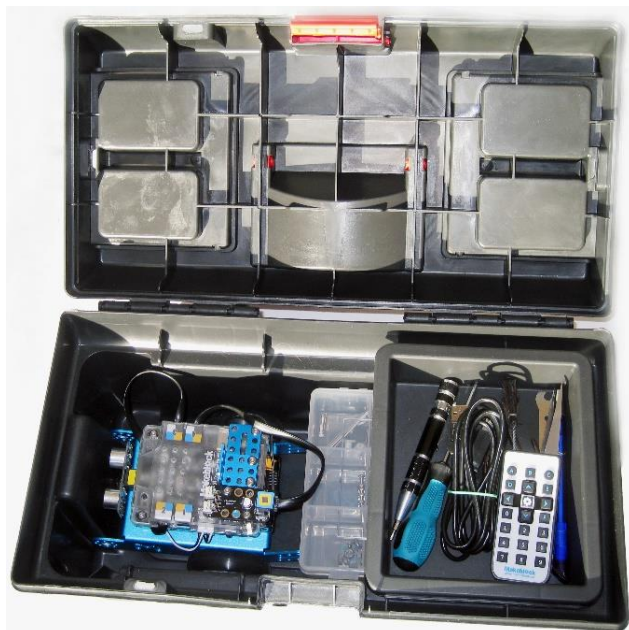
modification demands it I occasionally use a small white towel under mBot to catch any falling screws and nuts etc. These do bounce a long way if they are dropped and hit a hard surface - and on a patterned carpet, well ... The towel (not shown here) has slots cut in it to fit over the two brackets.

Additional Storage



I did note in my product overview in Chapter 3 (on page 10) that it's tricky to keep track of cables, spares and add-ons for tech toys and that is true with mBot too.

I also mentioned there that I did succumb to a bigger storage option and here it is - a small tool-box which cost about £6 from B&Q. As you can see from the picture on the right, mBot fits nicely into the bottom of the toolbox with space for two divided plastic boxes next to it. This particular toolbox has the small tray that you can see on the right, which is a perfect size for tools, cables & the IR remote.





Chapter 18 - Quo Vadis ?



After buying in all of the add-on components described in the appendices, and then constructing and programming each of the prescribed models; where to next? I had gained quite a lot of lovely Makeblock bits & pieces but what can I make with them? When I looked at my assembled mBot collection, (see below) nothing particularly came to mind.

I then considered what tasks mBot itself can do, and I decided that using one of its two primary outputs - the Ultrasonic (distance) Sensor made sense for developing a new project. I also decided that mBot's two motors powering the drive wheels, together with the two servos (gained as add-on components) are the only other things have any real value for creating robotics projects, apart from the amassed collection of beams, linkages, fixings and plates.

So, distance sensor, motors & servos ... what to create?



The Ultrasonic Sensor enables the mBot to 'see' and 'recognize' objects, avoid obstacles, measure distances or detect movement and uses the same scientific principle as bats; it measures distance by calculating the time it takes for a sound wave to hit an object and come back; just like an echo. The ultrasonic sensor has a (supposedly) maximum range of 400 centimetres (4M) - could I perhaps use mBots Ultrasonic Sensor to simulate the radar, sonar or depth devices on a ship? I began to research mBot models on the internet and found a *YouTube* video showing a very likable 'Air Traffic Control' radar simulator model based on mBot:
<https://www.youtube.com/watch?v=UjKnSca2tVs>

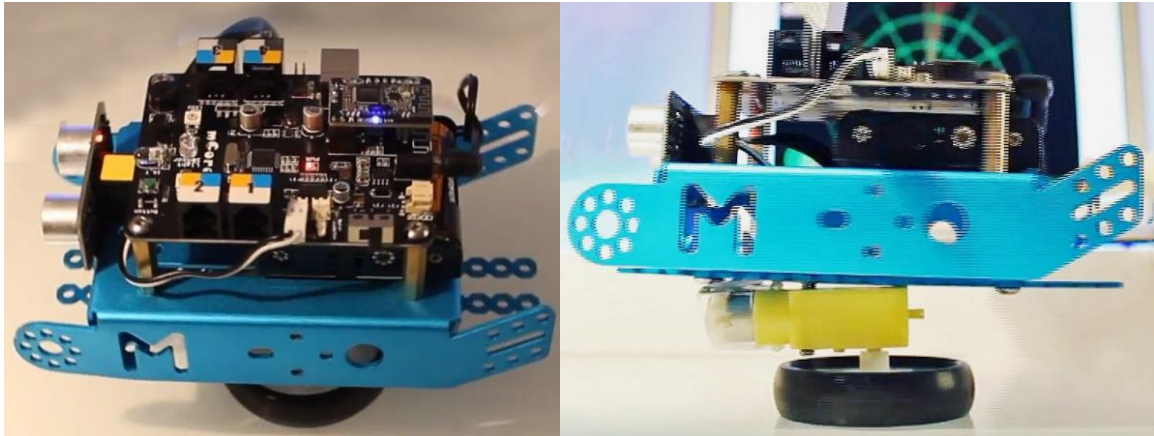
This was a creation of Javier & Eva Venegas who have a Spanish educational website: 'KidsandChips' - a site showing (for a variety of platforms and kits) educational robotics projects which are capable of being constructed and programmed by anyone. The site is written in Spanish, but it is translatable by Google (to see it, use the following link):

www.kidsandchips.es



mBot and Me *a Beginner's Guide*

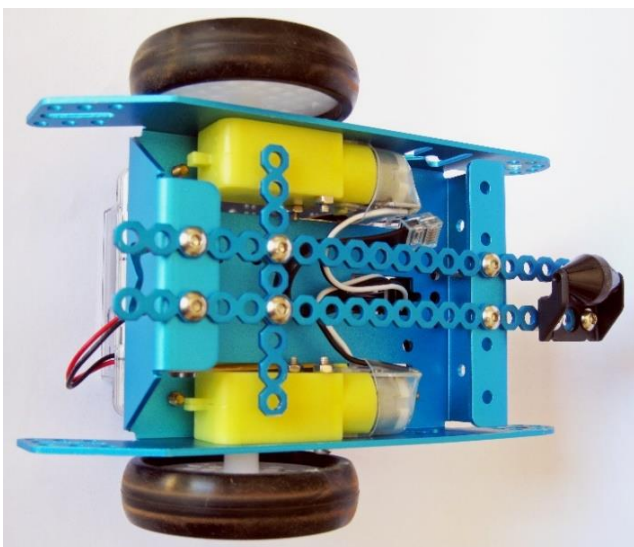
As much as I liked the *'KidsandChips'* radar simulator project, I really did not like the method they used to rotate mBot. This required the removal of the wheels and motors from mBot's chassis and then one motor and wheel fixed below mBot at the natural balance point. OK, it seems to work - but so vulnerable to damage since the whole model is balanced on the single (not over strong) integral axle of one motor. Makeblock do include spare motor shafts in the mBot kit - probably for good reason! Below are two views of this cut from the *'KidsandChips'* YouTube video - to my mind, a rather scary balancing act:



My first thought on modifying this project was to keep mBot more-or-less intact. In Appendix 13 (page 243) which outlines *'The Intelligent Desk Light'* project I indicated my reluctance to dismantling mBot by substituting mBots chassis for an aluminium ready-meal tray to make a lamp-shade.

Could I not make mBot rotate on the spot by driving one wheel forwards and the other wheel backwards? A brief test of this showed that mBot could indeed rotate, but it moved about a bit and did not remain on the spot.

I also found the motors supplied with mBot needed a speed setting of 30% to drive mBot happily; this gave mBot a rotation time of about 6 seconds per full revolution. What I needed I thought was a baseboard with an upright pivot spigot through it and a pivot hole on the underside of mBot in line with the two axles and midway across the chassis.



You can just see from the pictures above that the *'KidsandChips'* team used cuttable linkages bolted to the underside of mBots chassis to mount their motor.

I thought that I could do the same to create the required pivot point and you can see clearly in the picture on the left how this was achieved with one short strip of cut linkage bolted across the axle (pivot point) of the robot.

You can also see that that I had to remove the line-following sensor and the front wheel to utilise the threaded holes on the chassis - as a bonus however this allowed me to reposition the front wheel at 90° to its normal position allowing it to follow a circular track around the pivot easily and with minimal drag.

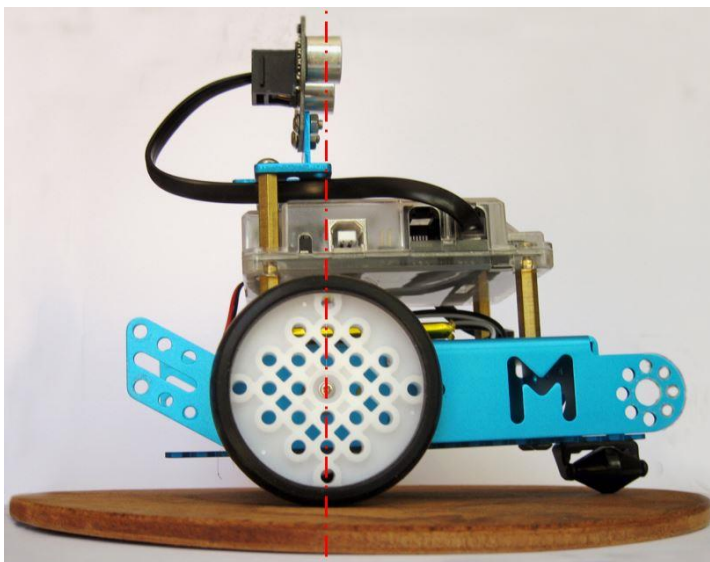


All that I had to do was to create a circular baseboard from some scrap plywood. A diameter of 225mm seemed ample to allow mBots front roller-ball wheel to be supported.

After measuring the height of my new pivot point on the underside of mBots chassis (25mm), I added a 30mm countersunk headed screw to the centre of my circular baseplate (see the picture of this on the right);



I positioned mBot with its new pivot point fitted over the central pivot screw on the baseplate disk and was ready for a second test. It worked as expected - a perfectly smooth rotation and the front roller-ball wheel equally smoothly - a very successful experiment!



My next modification (shown here on the left) was to reposition the ultrasonic sensor on top of mBot – vertically in line with the rotation point.

This was so easy to do and as mBot rotated the sensor maintained position over the rotation point. My modification still looked essentially like mBot too, but this had now become a model that *could* simulate a 'rotating radar' unit!

I have found that I can keep the three pieces of cut linkages underneath mBot most of the time.

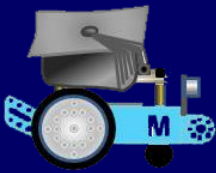
Since I used two M4 wing-nuts to hold the front roller-ball wheel in place it takes seconds to rotate it back in line with the robot (with only one bolt and wing-nut needing to be repositioned). The sensor can be quickly refitted back in its normal position on the front of mBot too and since I hardly ever use it the line-follower module is not usually re-attached to my mBot.

Another mBot based project on the 'KidsandChips' website also attracted my interest. This was another YouTube video of a simple mBot powered 'Spirograph' drawing machine; it was so intriguing that it inspired me to start an mBot 'drawing machines' project. You can find this inspiration for yourself at:

<https://www.youtube.com/watch?v=-Biq2Hys-5w>

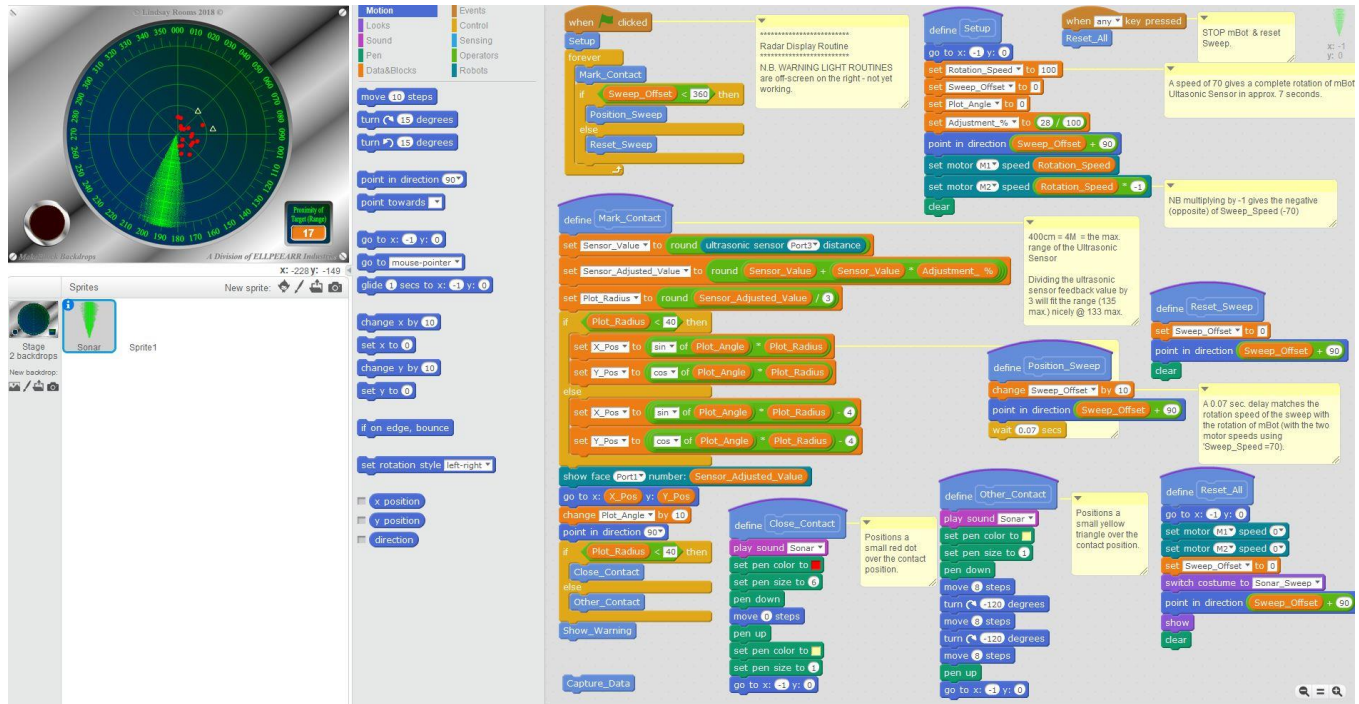
One particular project on yet another Spanish website shows in quite some depth, robotic sensing and logical decision making in getting mBot to escape from a labyrinth. The line-follower sensor and ultrasonic sensor that come with mBot are the only sensors needed to implement an outstandingly good and well documented maze-solving algorithm. This website contains much valuable information and it is authored by Dani Sanz, a professional who specialises in on-line courses in educational robotics. I may well try out his ideas too - the link to his work is:

<https://juegosrobotica.es/>



Chapter 19 - An mBot 'Radar' Simulation Project

Following on from my experiments with a rotating-on-the-spot mBot (describe in the last chapter) I got to work and eventually created a working version of a radar screen simulation using mBlock 3 - this was about six months before I switched to mBlock 5.0.1. A screenshot of the mBlock 3 screen running this project is shown below:



Eventually, I got back to recreating this project using mBlock 5. This was a little harder to achieve since I had to juggle robotics scripting on the 'Devices' tab and stage graphics programming on the 'Sprites' tab. I had to wait another four months before I had developed my understanding of mBlock 5 enough to be able to rewrite this.

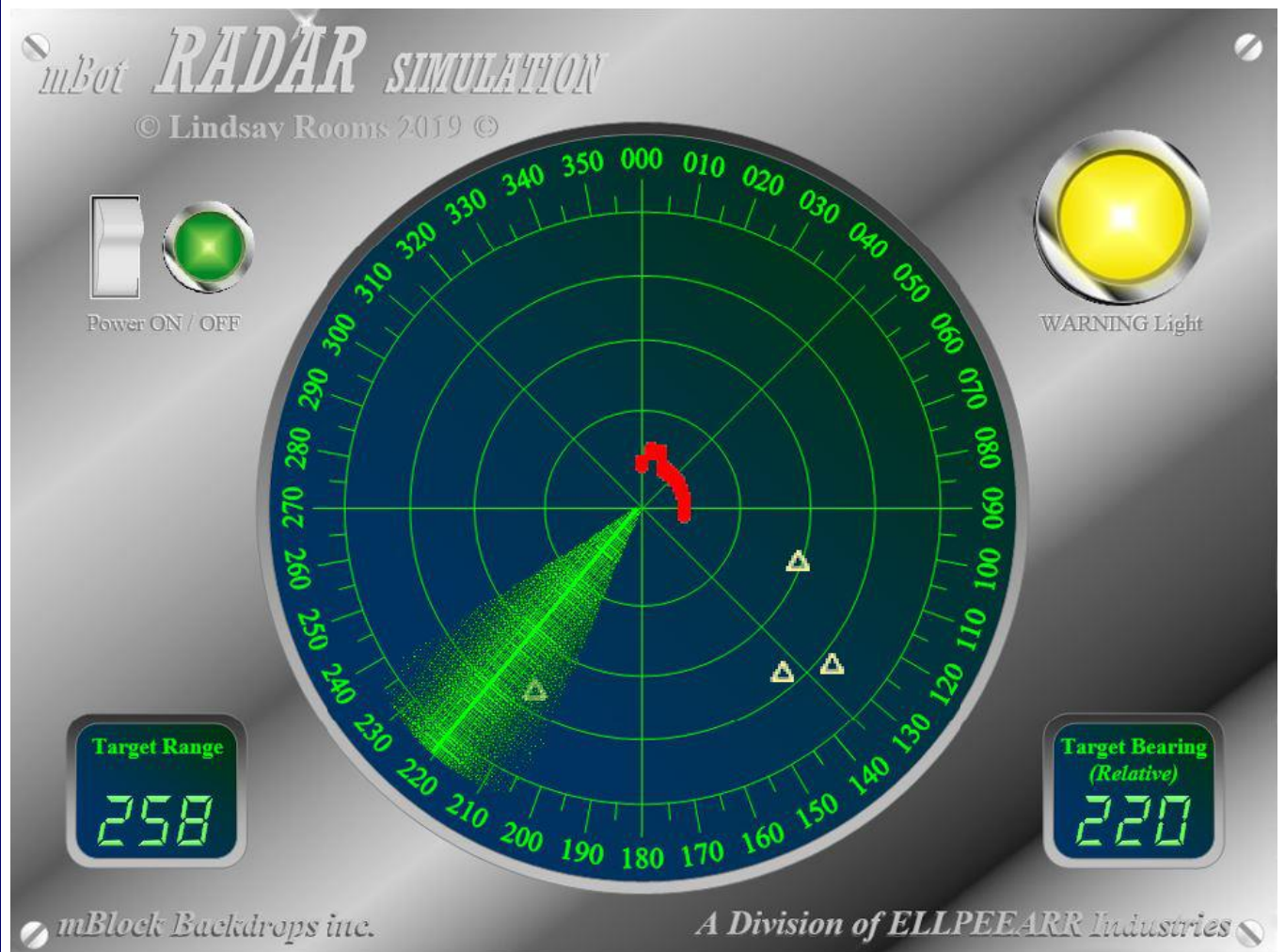
At its simplest, this project needs mBot to rotate on its own axis (with the ultrasonic sensor mounted vertically above that axis - *as shown on the previous page*) and the x and y coordinates of the position of any identified objects around mBot then calculated from two pieces of data.

The first needs the sensor to be repeatedly polled and the data feedback transmitted to Mblock 5 whilst the second needs to be estimated by timing (matching as closely as possible mBot's rotation speed).

1. The distance from any object identified (*Range*).
2. An estimation of the angle of mBot's rotation (*Bearing*).

Using these two pieces of data and some simple trigonometry, the x, y coordinate positions of any objects identified around mBot can be identified by processing the following two formulae:

$$\text{Position in X} = (\text{Sin of Bearing}) \times \text{Range}$$
$$\text{Position in Y} = (\text{Cos of Bearing}) \times \text{Range}$$



Using mBlock 5's potential to link and interact with feedback from mBot's sensors via 'stage' graphics enables you to create the *illusion* of a 'Radar Screen Display'. The project backdrop that I created is shown above. This has on top of it a rotating analogue 'Sweep' graphic, which moves in sync. (*more-or-less*) with mBot's on-the-spot rotations and using the basic mathematical principles outlined on the previous page, the identified position of any object around mBot can be used to position a pen-drawn 'Radar Contact' graphic on top of the radar display.

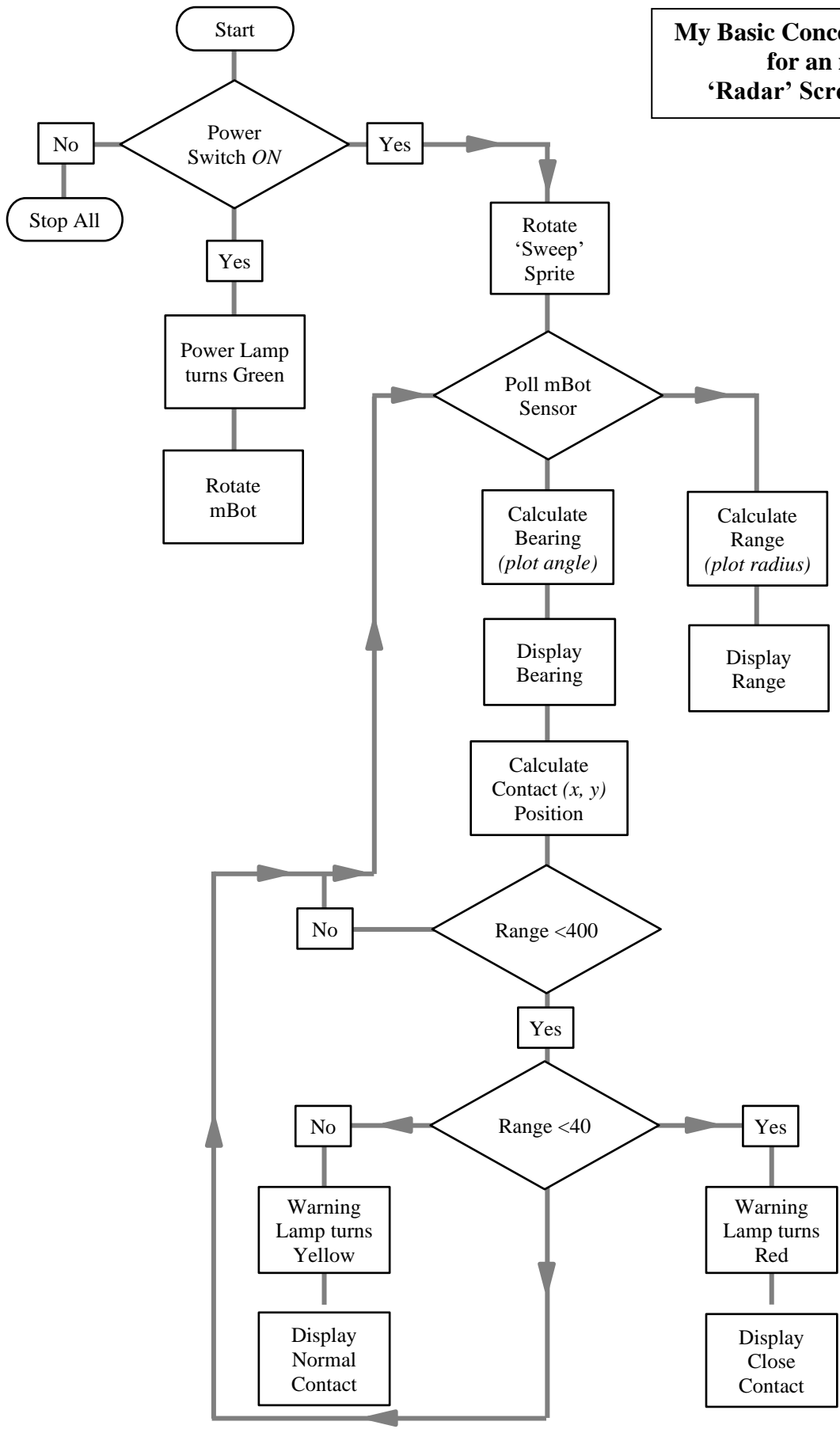
The sweep indicates the bearing direction being polled and the 'Target Bearing' window shows this numerically too. The 'Target Range' window shows the current reading from mBot's Ultrasonic Sensor. N.B. These two windows use the graphical LED digits used in my 'Control Interface' project (see Chapter 19).

I began this project (initially using mBlock 3) by testing the ultrasonic sensor. I pointed mBot at a wall and using a tape-measure to position it noted the sensor feedback readings at different distances. Rather disconcertingly all readings were 27 - 28% less than the distance indicated by mBot's position on the tape-measure. I also noted that the sensor values constantly fluctuate slightly if the sensor is detecting something, but the sensor always returns 400 if it is not detecting anything (its out-of-range reading!).

On the next page is the algorithm that I devised to work out the sequence of how the project might work:



My Basic Concept Algorithm for an mBot 'Radar' Screen Project





As outlined earlier, the problems with using the ultrasonic sensor are twofold. Firstly, my ultrasonic sensor under-reads by about 27% (this may vary slightly for individual Me sensor modules?).

Secondly, the point at which the sensor stops reading distances (its range) varies according to the ambient light around it and the range of the sensor is nowhere near the 400cm (4M) range expected. I guess that each individual Me sensor might vary slightly in this too.

- In bright daylight (outside) my sensor monitor reading did not start to fluctuate from a fixed 400 until it was below 150cm (1.5M) - an extremely poor result!
- Indoors (in reasonable daylight) the sensor monitor started to fluctuate at 230cm (2.3M) and with curtains drawn (performing in subdued light) it still only started to perform at 270cm (2.7M)
- Testing it at night, in darkness, the sensor monitor readings begin to fluctuate at 330cm (3.3M) - which corrected by adding 27% gave a max. range of 4,2M.

Matching the on-screen radar sweep graphic's speed with mBot's rotational speed is hard to achieve with any degree of accuracy, so as usual, I carried out several 'test-bed' exercises first with both the graphic rotation and the speed of mBot.

I discussed motors in Chapter 15 (on page 125). Individual motors will vary in output to but generally are only reliable at speed settings of over 30% (76.5/255). At speeds lower than this, mBot's motors tend to stall. This speed (the slowest I can reliably set) will rotate mBot clockwise in approx. 6 secs per revolution. Carrying out programming tests relating to rotating the radar sweep, I settled on making it move in 10° increments which still looked fairly smooth in action. I also found that to match mBot's rotation speed I had to build in to the rotation script a delay (wait) time of 0.13 secs per 10° of revolution of the graphic for it to match mBot.

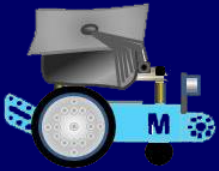
N.B. $0.13 \times 36 = 4.7 \text{ secs}$; but even though ($0.165 \times 36 = \text{the required } 6 \text{ secs}$) a wait time of 0.13 secs worked best in my tests. You might (& probably should) test both your own motor speeds and measured readings from your own ultrasonic sensor.

Experimenting with a 'Sonar Ping' sound file was valuable too. Adding one which lasted just under 2 seconds added much to the illusion but needed playing 'until done' rather than just 'started', which cropped the sound into a short blip.

Once I had established the basic principles of what I wanted to achieve it was time to begin programming the project in mBlock 5. On the 'Sprites Tab' I created the following eleven sprites:

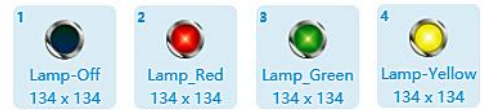


The two 'Lamp' sprites are named 'Lamp_On-Off' and 'Lamp_Warning'. The three 'Bearing' and three 'Range' sprites are appended with '_Hundreds', '_Tens' and '_Units' in exactly the same way as they were in my Control Interface project (see Chapter 15). I had saved one set of my LED sprites from that project as a .sprite3 file and I was now able to imported it into this project and duplicate it; which saved much time and effort!

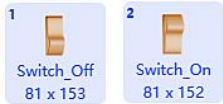


mBot and Me a Beginner's Guide

The two 'Lamp' sprites need the same four costumes added to each of them (so just make one sprite & then duplicate it!).



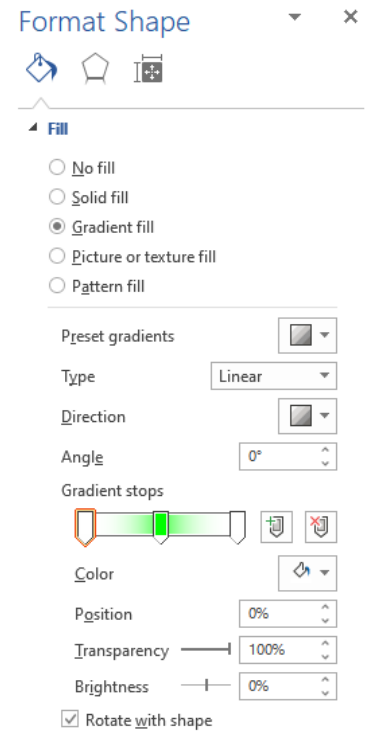
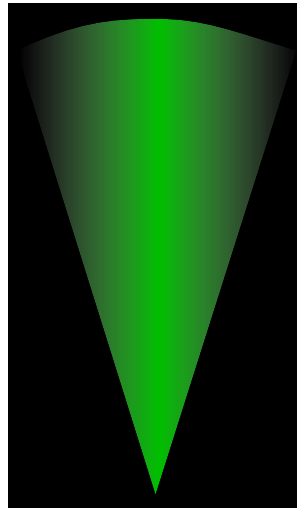
The 'Contact' sprite contains NO graphic costumes as such, but to exist, it *must* have a blank costume that cannot be seen on the stage. This sprite is just a neat way of containing all of the scripts require to position, draw and ping any identified 'radar' contacts.



The 'Switch' sprite has two costumes, 'Switch_Off' and 'Switch_On'.

Whilst the 'Sweep' sprite only has just the one costume shown here on the right.

To create this in Word, I drew a 'Freeform Shape' and filled it with the vivid green colour (R 90, G 255, B 60). I then formatted the Shape with a 'Gradient' fill as shown on the right.



I then followed the method (described in Chapter 14) to create a .png sprite file; very carefully turning all of the white background transparent using Photoshop's 'Magic Eraser' tool. It took some time to make this work as intended.

The LED sprites had all of their costumes imported with them when I loaded my .sprite 3 file. The first sprite that was imported, was duplicated five more times to create the six LED sprites required for this project.

From now on, I intend to provide the bare minimum of explanation, but I hope that I will have provided you with enough clues on the following pages to enable you to create this project.

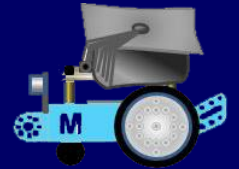
After setting up the sprites required, I created the following Broadcast message names:

Check_For_Contact, Motors_Run, Motors_Stop, Poll_Sensor, Run_All, Send_Data, Set-Up_All, Stop_All, Switch_Lamp, Update_Digits, Warning_Off, Warning_Set, Zero_Digits

I also created the following Variables:

Bearing_Digits_Shown, Bearing_Hundreds, Bearing_Tens, Bearing_Units, Bearing_Xpos, Bearing_Ypos, Digits_Space, mBot_Speed, Plot_Angle, Plot_Radius, Range_Digits_Shown, Range_Hundreds, Range_Tens, Range_Units, Range_Xpos, Range_Ypos, Sensor_Adjustment, Sensor_Value, Sweep_Offset, Sweep_Speed, Switch_Pos, Target_Range, X_Pos, Y_Pos

On the next page is a screenshot of all of the 'DEVICES' tab scripts. These are required to operate mBot and to provide feedback back to the 'stage'. Only one script is required to start mBot's motors. Adding +2 to the motor power on port M2 corrects a slight difference in speed between the motor on port M2 and the marginally faster motor on port M1 - you may need to make a similar adjustment. *N.B. Using 'anticlockwise' for each motor turns mBot to the right - clockwise!*



Scripts created on the 'Devices' tab:

```

when I receive Motors_Run
  DC motor motor port1 anticlockwise rotates at power mBot_Speed %
  DC motor motor port2 anticlockwise rotates at power mBot_Speed + 2 %

when I receive Motors_Stop
  stop moving

when I receive Poll_Sensor
  forever
    set Sensor_Value to round ultrasonic sensor port3 distance cm
    set Target_Range to Sensor_Value + round Sensor_Value * Sensor_Adjustment
    set Plot_Radius to round round Target_Range / 3.2
    broadcast Send_Data

define Bearing_Update
  set Bearing_Digits_Shown to length of Plot_Angle
  set Bearing_Hundreds to letter Bearing_Digits_Shown - 2 of Plot_Angle
  set Bearing_Tens to letter Bearing_Digits_Shown - 1 of Plot_Angle
  set Bearing_Units to letter Bearing_Digits_Shown - 0 of Plot_Angle

define Range_Update
  set Range_Digits_Shown to length of Target_Range
  set Range_Hundreds to letter Range_Digits_Shown - 2 of Target_Range
  set Range_Tens to letter Range_Digits_Shown - 1 of Target_Range
  set Range_Units to letter Range_Digits_Shown - 0 of Target_Range

when I receive Send_Data
  if Plot_Angle = 360 then
    Bearing_Reset
    Range_Reset
  else
    Bearing_Update
    Range_Update
  broadcast Update_Digits

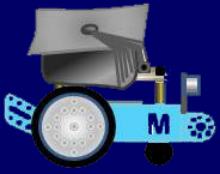
when I receive Zero_Digits
  set Bearing_Hundreds to 0
  set Bearing_Tens to 0
  set Bearing_Units to 0
  set Range_Hundreds to 0
  set Range_Tens to 0
  set Range_Units to 0
  broadcast Update_Digits

define Bearing_Reset
  set Plot_Angle to 0
  set Bearing_Hundreds to 0
  set Bearing_Tens to 0
  set Bearing_Units to 0

define Range_Reset
  set Sweep_Offset to 0
  set Range_Hundreds to 0
  set Range_Tens to 0
  set Range_Units to 0
  
```

On the next page are the 'Switch' sprite scripts created on the 'Sprites' tab. The 'switch' (activated by the 'when this sprite clicked' hat block) sets up all required variable values and starts both mBot and the 'Radar Screen' graphic.

N.B. Full-screen 'Presentation mode' needs to be selected but there is no way of automating this feature.



mBot and Me a Beginner's Guide



There are three critical Variables values to be set-up here.

(1) Setting 'mBot_Speed' to 30% will make mBot's motors work at a speed of $76.5 / 255$. At speeds lower than this, mBot's motors tend to stall!

(2) Setting a 'Sweep_Speed' timing interval of 0.13 will give a 6 second rotation time for the sweep graphic (which matches the time it takes mBot to complete one revolution).

(3) Setting the 'Sensor_Adjustment' to 27/100 (27%) gives a sensor feedback value equivalent to the actual distance targeted.

```
when this sprite clicked
  broadcast Set-Up_All
  broadcast Run_All
  go to x: -199 y: 90
  start sound Click
  if Switch_Pos = 0 then
    Switch_On
  else
    Switch_Off
```

```
define Switch_On
  switch costume to Switch_On
  show
  broadcast Motors_Run
  broadcast Switch_Lamp
  set Switch_Pos to 0
```

```
when I receive Set-Up_All
  set mBot_Speed to 30
  set Sweep_Speed to 0.13
  set Sweep_Offset to 0
  set Plot_Angle to 0
  set Sensor_Adjustment to 27 / 100
```

```
define Switch_Off
  switch costume to Switch_Off
  show
  broadcast Stop_All
  broadcast Switch_Lamp
  set Switch_Pos to 0
  broadcast Zero_Digits
  wait 0.1 seconds
  stop all
```

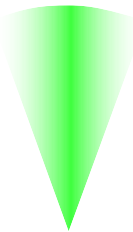
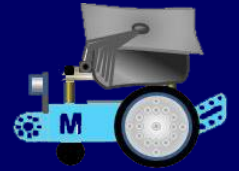
Shown below are the three short 'Lamp_On-Off' sprite scripts, which are activated by the movement of the 'switch' sprite broadcasting the 'Switch_Lamp' message.

```
when I receive Switch_Lamp
  if Switch_Pos = 0 then
    Lamp_Off
  else
    Lamp_On
```

```
define Lamp_On
  go to x: -165 y: 90
  switch costume to Lamp_Green
  show
```

```
define Lamp_Off
  go to x: -165 y: 90
  switch costume to Lamp-Off
  show
```





Shown on the right are the two 'Sweep' sprite scripts, 'Run_All' and 'Stop_All'.

There are also two additional scripts here, 'when (down arrow) key pressed' and 'when (up arrow) key pressed'.

I used these to start & stop mBot manually when I was testing early versions of the project. You may find it useful to have them too.

```

when I receive Run_All
  forever
    broadcast Check_For_Contact
    if Sweep_Offset = 0 then
      erase all
    if Sweep_Offset < 360 then
      Sweep_Rotate
      wait Sweep_Speed seconds
    else
      Sweep_Reset

when I receive Stop_All
  broadcast Motors_Stop
  stop other scripts in sprite
  Sweep_Reset

when down arrow key pressed
  broadcast Motors_Stop
  stop other scripts in sprite
  Sweep_Reset

when up arrow key pressed
  broadcast Motors_Run

define Sweep_Reset
  set Sweep_Offset to 0
  point in direction Sweep_Offset + 90
  broadcast Warning_Off
  erase all

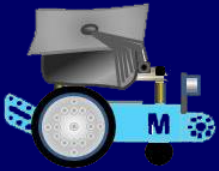
define Sweep_Rotate
  change Sweep_Offset by 10
  point in direction Sweep_Offset + 90
  
```

On the next page are all of the 'Contact' sprite scripts. You can probably see why they required a sprite to themselves even though a sprite graphic is not needed, only the transparent (invisible) one mentioned earlier. I tried experimented with using sprite costumes and stamping them on the stage display for this project but using the pen to create the contact graphics provided a much neater solution.

A critical addition to the main 'Check for Contact' script is checking if the 'Sensor_Value' is not in range (showing 400) and then exiting the script if true. The 'Set_Target_Pos' block script does the maths described at the beginning of this chapter to convert the sensor feedback into a contact position.

N.B. The centre of my 'radar' display screen is positioned at x 0, y -8, so adding -8 to 'Ypos' (in 'Set_Target_Pos') allows for this; and since the display has a diameter of 240 pixels then the script also sets the 'Max_Plot_Radius' to 120.

The 'Adjust_Position' self-defined block script offsets the start position for the drawn triangle, enabling it to be positioned evenly over the target position whilst the 'Ping_Contact' block script contains the method I eventually devised to play the two second 'Sonar' sound successfully.



'Contact' sprite scripts:

Contact

```
when I receive Check_For_Contact
broadcast Poll_Sensor
Set_Target_Pos
broadcast Send_Data
change Plot_Angle by 10
if Sensor_Value < 400 then
  Ping_Contact
  if Plot_Radius < 40 then
    go to x: X_Pos y: Y_Pos
    Contact_Closing
  else
    go to x: X_Pos y: Y_Pos
    Contact_Normal
```

```
define Ping_Contact
if Plot_Angle = 10 then
  play sound Sonar until done
if Plot_Angle = 60 then
  play sound Sonar until done
if Plot_Angle = 120 then
  play sound Sonar until done
if Plot_Angle = 180 then
  play sound Sonar until done
if Plot_Angle = 240 then
  play sound Sonar until done
if Plot_Angle = 300 then
  play sound Sonar until done
```

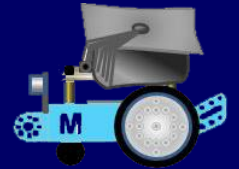
```
define Draw_Triangle
point in direction 90
set pen color to yellow
set pen size to 1
pen down
move 8 steps
turn 120 degrees
move 8 steps
turn 120 degrees
move 8 steps
pen up
```

```
define Contact_Closing
broadcast Warning_Set
set pen color to red
set pen size to 6
pen down
move 0 steps
pen up
```

```
define Set_Target_Pos
if Plot_Radius > 120 then
  set Plot_Radius to 120
set Plot_Radius to Plot_Radius
set X_Pos to sin of Plot_Angle * Plot_Radius
set Y_Pos to cos of Plot_Angle * Plot_Radius - 8
```

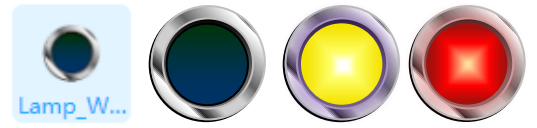
```
define Contact_Normal
broadcast Warning_Set
Adjust_Position
Draw_Triangle
```

```
define Adjust_Position
point in direction -90
move 4 steps
point in direction 180
move 2.6 steps
```



Shown below are the 'Lamp_Warning' sprite scripts.

N.B. The 'Warning_Off' script had to be a 'Broadcast' message and not a self-defined block so that it could be called by the switch sprite to set the warning lamp to show the 'Lamp_Off' costume.



```

when I receive Warning_Set
  if Plot_Radius < 40 then
    Warning_Red
    wait 0.15 seconds
  if Plot_Radius > 40 then
    Warning_Yellow
    wait 0.15 seconds
  if Sensor_Value > 399 or Switch_Pos = 0 then
    broadcast Warning_Off
    wait 0.15 seconds

when I receive Warning_Off
  go to x: 180 y: 100
  switch costume to Lamp-Off
  show

define Warning_Yellow
  go to x: 180 y: 100
  switch costume to Lamp-Yellow
  show

define Warning_Red
  go to x: 180 y: 100
  switch costume to Lamp_Red
  show
    
```



Next, are the sets of scripts for the three individual sprites required to display the three digits of the 'Bearing' (angle of the 'Sweep') using the LED digits from my graphics library.

As mentioned earlier, one set of these was saved as **.sprite3** file and this file containing all of its costumes and its scripts was imported into this project.

On the right are the 'Bearing_Units' sprite scripts'. N.B. the 'Update_Digits' script only required minor modifications to the imported original.

```

when I receive Update_Digits
  Digit_Pos
  go to x: Bearing_Xpos y: Bearing_Ypos
  if Bearing_Units < 1 then
    switch costume to Digit 0
  else
    switch costume to Bearing_Units
  show

define Digit_Pos
  set Bearing_Xpos to 201
  set Bearing_Ypos to -128
  set Digits_Space to 15
    
```




mBot and Me a Beginner's Guide

Below is the single script required for the 'Bearing_Tens' sprite and slightly below that, the single script required for the 'Bearing_Hundreds' sprite.

```

when I receive Update_Digits
go to x: Bearing_Xpos - Digits_Space y: Bearing_Ypos
if Bearing_Tens < 1 then
  switch costume to Digit 0
else
  switch costume to Bearing_Tens
show

```

Finally, to complete this project, you need a similar set of LED sprites to show the actual

```

when I receive Update_Digits
go to x: Bearing_Xpos - Digits_Space * 2 y: Bearing_Ypos
if Bearing_Hundreds < 1 then
  switch costume to Digit 0
else
  switch costume to Bearing_Hundreds
show

```

distance from mBot's ultrasonic sensor to any identified target. Once I had completed the scripts for all three of the LED digit sprites shown above, I duplicated each of them and renamed them as 'Range ...' sprites. Shown below are the two modified scripts for the 'Range_Units' sprite and

below that, and shown (slightly smaller) are the modified single scripts for the two remaining sprites.

```

when I receive Update_Digits
Digit_Pos
go to x: Range_Xpos y: Range_Ypos
if Range_Units < 1 then
  switch costume to Digit 0
else
  switch costume to Range_Units
show

```

'Range_Units' sprite

```

define Digit_Pos
set Range_Xpos to -170
set Range_Ypos to -128
set Digits_Space to 15

```

I hope that you have fun with the complexities of this project.

N.B. It is meant to test you on what you have learned so far!

```

when I receive Update_Digits
go to x: Range_Xpos - Digits_Space * 2 y: Range_Ypos
if Range_Hundreds < 1 then
  switch costume to Digit 0
else
  switch costume to Range_Hundreds
show

```

'Range_Hundreds' sprite

```

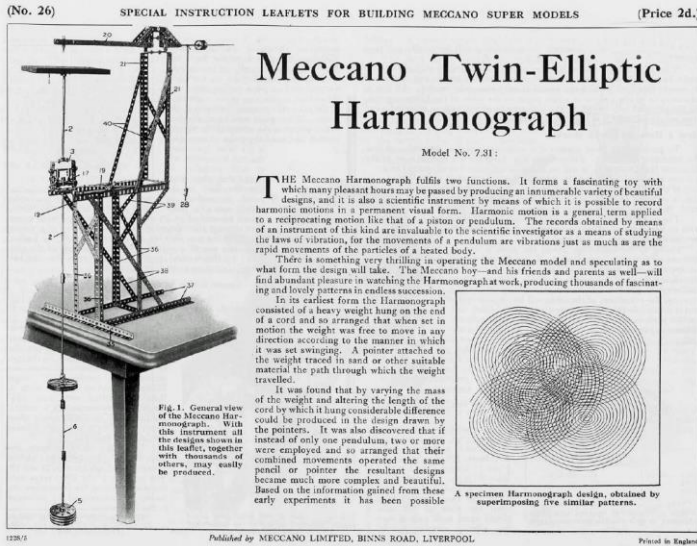
when I receive Update_Digits
go to x: Range_Xpos - Digits_Space y: Range_Ypos
if Range_Tens < 1 then
  switch costume to Digit 0
else
  switch costume to Range_Tens
show

```

'Range_Tens' sprite



Chapter 20 - An mBot 'Drawing Machine' Project



no copyright infringement is intended with the use of this image

It must be over sixty years ago when I first saw a picture of and read about a 'Harmonograph' - I tried to make one using 'Meccano', but it didn't work, but back then I was only ten! After having seen the 'KidsandChips' mBot powered version which I described in Chapter 18 (on page 159), I decided that it was long overdue for me to have another go at a drawing machine. The 'KidsandChips' version described as a 'Spirograph' uses one of mBots motors and drive wheels to rotate the paper whilst the other motor and wheel uses a simple crank to drive a couple of linkages to push-and-pull a pen backwards and forwards over the rotating paper.

N.B. A 'Spirograph' actually uses meshed gear wheels to generate the graphics, which this model does not. I decided that I would attempt to make several mBot variants of the most commonly found drawing machine types that use a 'crank-arm' to convert rotary motion into linear motion. These will require hardly any mBlock scripting (other than that needed to start and stop the motors). However, they will need virtually all of a working mBot to be dismantled to redeploy the parts required. These models will also need many of the additional Makeblock components I have acquired (from add-on packs) too.

Some historical facts:

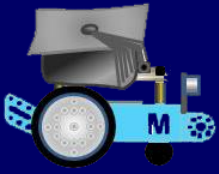
The '**Pantograph**' (its earliest form was for copying writing and dates back to the early 1600's) is a mechanical linkage based on parallelograms so that the movement of one point or pen, in tracing an image, produces identical movements in a second point or pen. If a line drawing is traced by the first point, an identical, enlarged, or miniaturized copy will be drawn by a pen or other device fixed to the second point for duplication, sculpture, minting, engraving or milling. In 1827 an English architect and engineer, Peter Desvignes developed a machine to create elaborate spiral drawings intended to prevent bank note forgeries. He called this a '**Speiragraph**'.

In the mid-19th century, the '**Harmonograph**' (which cannot conclusively be attributed to a single person) used swinging pendulums to create geometric images. One pendulum moved the pen back and forth along one axis whilst the other pendulum moved the drawing surface back and forth along the other axis. By varying the frequency and phase of the pendulums relative to one another different patterns could be created to create ellipses, spirals, figures-of-eight and complex Lissajous curves. Some complex harmonographs also involved rotary motion too.

In 1908, '**The Marvellous Wondergraph**' gear-based drafting toy was advertised for sale.



no copyright infringement is intended with the use of this image



mBot and Me a Beginner's Guide

In 1965 Denys Fisher first exhibited his very successful variation on this theme, a toy which he called *'Spirograph'*. *This has now become the ubiquitous generic term for the generation of graphics using mathematical 'roulette' curves technically known as hypotrochoids and epitrochoids.*

With some simple programming in mBlock 5 you can generate a variety of patterns of this nature and you can find many examples of such scripts if you look for them in the Scratch Wiki. - (shown here on the right is one simple example of this):

Most programmes like this one rely on drawing a polygon on the stage using pen commands and then moving the origin, setting the amount of rotation and number of repeats; & often changing pen colour too.

In this example you can set the value of the variables in the blocks on the right of the diagram by changing the value in the block bubble and then clicking the block before clicking the green flag.

```
when clicked
hide
go to x: 0 y: 0
point in direction 90
erase all
repeat Shape_Repeat
  repeat 10
    pen down
    change pen color by 10
    set pen size to Pen_Size
    move 50 steps
    turn 360 / Polly_Sides degrees
    turn 360 / Shape_Repeat degrees
set Polly_Sides to 5
set Pen_Size to 1
set Shape_Repeat to 10
```

This *'programming only'* solution however does not make use of your mBot at all and **my aim was to create mechanical drawing machines which could draw with a real pen on real paper.**

I experimented with several variants of these machines, the first one very much simpler than the *'KidsandChips'* version which had inspired me to start this project. Both of these use a moving arm or arms operated by a crank to manipulate the pen and they also rotate the paper which generates petal-like lobes. These types are generally known as 'roulette' drawings and are bounded by a circle, just like a 'Spirograph'. As hinted-at on the previous page, your mBot needs dismantling to provide you with the required components.

This did not take long. I removed the drive wheels, the motors, the line-follower module and the front mini-caster wheel, leaving the ultrasonic sensor module in place and the mCore board inside its protective casing (its cover) still secured to the chassis by, and supported on, four hexagonal 25mm brass spacer pillars.

Despite my reluctance to destroy a working mBot robot; it is actually very rewarding to have all of its components available (together with any other parts accrued from add-on component packs) to enable you to experiment with mechanisms.



I had in mind what I wanted to try first and it took me about five hours of happy fiddling to come up with a working solution. It's creation was quite literally a 'top-down' approach to model-making by first working out how paper could be attached to one of the motors (using one drive wheel) whilst the other motor and second drive wheel provided a crank-drive to move a pivoting arm. The tyres were then removed from the wheels. Without the tyres, the drive wheels are 58mm in diameter.

How to hold-down and support the paper on top of one of the drive wheels? I needed to fix something to the gear wheel - cardboard, too bendy; plywood OK but heavy; so probably some form of plastic then? All of the images generated by this device would be circular. So circular in shape, what could I use?

I had a bright idea and sat an old Compact Disk (CD) on top of the drive wheel. It looked to be (at 120mm dia.) slightly too small, but it seemed a very neat and lightweight solution. I found an old erasable/recordable CD I had used years earlier to store computer data and peeled off the sticky label that I had added to it. This produced an unexpected bonus in that most of the shiny reflective recording surface came off with it. After a few rubs with an abrasive washing-up pad the remainder came off too leaving me with a very neat clear (with a blueish tinge) plastic disk, marginally scuffed by the scrubbing, but perfect for the job.

I carefully measured across one of mBot's drive wheels and the outer ring of four holes seemed to be spaced at 48mm dia. centres. I used a pair of odd-leg callipers to mark a circle 48mm dia. on to the CD. I put the gear wheel on top to check that I could see the circle through all four holes and clamped the wheel down on to the CD to use it as a drilling jig to drill four 4mm holes through it.



I probably could have bought some countersunk headed screws to make a neater job of bolting the CD to the drive wheel but decided to use four of the 14mm x M4 screws that I had as part of my mBot component collection.

The pan-heads of these were slightly raised (about 2mm) above the surface of the CD - and the CD (as they do) had a hole in the middle, so I now needed to create a flat surface above the screw heads. I decided on making a covering disk of two layers of 2mm thick cardboard (this was just about cuttable into 120mm circles with sharp scissors).



Using my drilled CD as a template, I drilled through one circle of cardboard (the bottom layer) with a 4mm drill using some old plywood as a support. Removing the CD, I then used an 8mm dia. flat bit to drill through each hole in the cardboard again to widen them. I then glued the this disk to the second cardboard disk and as you can see in the image on the left, the holes in the bottom layer create a recess allowing the heads of the screws to be covered by the top layer.

I bolted the CD on to the drive wheel and intended to use 'Glu Dots' (thin sticky pads) to attach the two-layer cardboard sandwich to the CD, but I couldn't yet - not until after the drive wheel has been attached to the spindle of the motor using its central self-tapping screw.



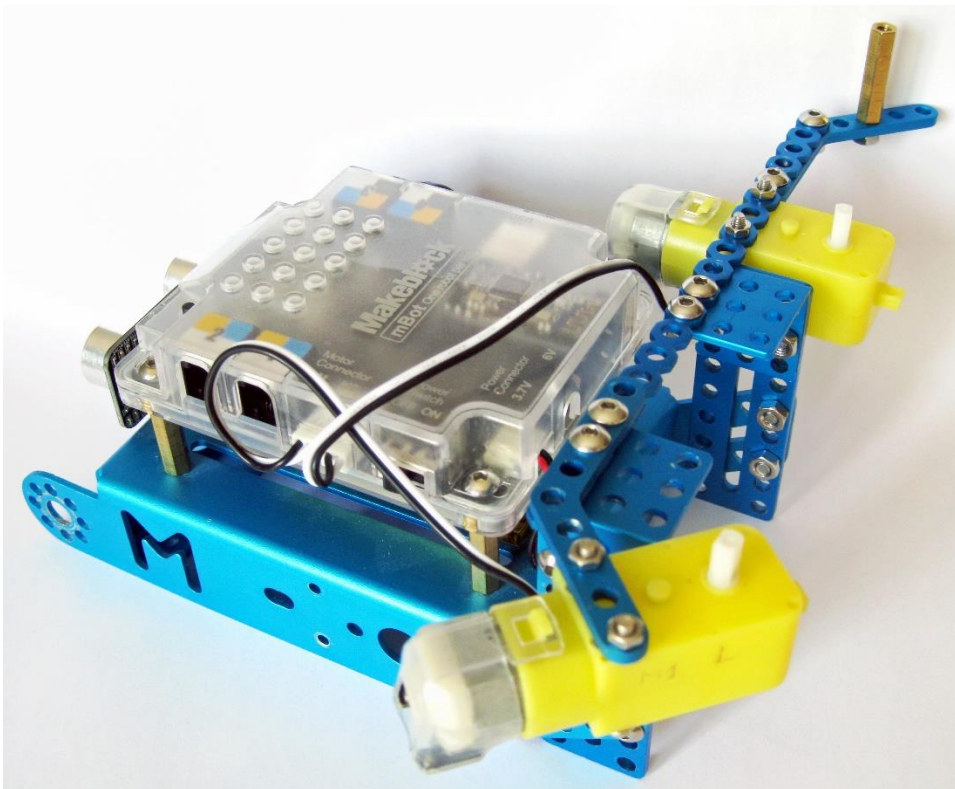
mBot and Me *a Beginner's Guide*

I was pleased with this CD and cardboard sandwich assembly which didn't seem to add much to the overall weight of the drive wheel once it was attached to the paper-drive motor. My plan for adding drawing paper to the machine was to attach individual disks of paper to the flat cardboard top surface using 'Glue-Dots' as required (and it is very easy to draw around another old CD, fitting three 120mm circles on to one A4 sheet of paper and then cutting them to shape).

The second problem was how and where to mount the motors bearing in mind that the cable from each motor is only approx. 160mm long; and to obtain power they need to reach to the M1 & M2 motor connection sockets in the middle of the left side of the mCore circuit board. This means mounting the motors to the rear of mBot's chassis and at about the same height as the mCore board; unless the mCore board is detached from its usual position on top of the chassis.

N.B. It is quite hard to extract the tight-fitting plugs on the motor cables from their sockets so it's quite important to use a small pair of 'snipe-nose' pliers to hold them firmly whilst removing or locating them into the tiny sockets on the mCore board.

The two drive motors have just two mounting holes, 2,5mm dia. at 17mm centres apart and use two M2.5mm x 25mm screws to attach them to either side of mBot's chassis. Trying to attach them to standard Makeblock components (plates, beams & linkages) is not straightforward since these all have 4mm dia. holes at 16mm centres! I was tempted to try increasing the holes through the motor casings very slightly but didn't dare to in case the motors were damaged; but with a bit of fiddling, the two screws will *just* pass through every other hole in a beam or linkage. Not, good, but it works!

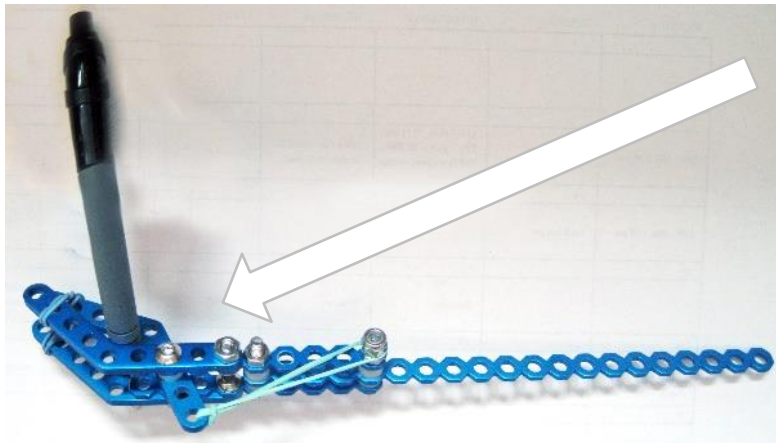
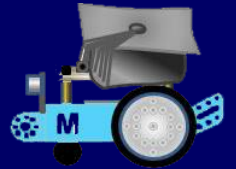


You should have enough information from the next few pictures to enable you to build something similar.

On to the back of mBot's chassis I built the motor support arm. This looks like the picture on the left. Keeping mBot as it is, really helps to provide a solid and steady base to absorb the movements generated by this project

The 25mm Brass pillar is for the pivot point screw; and as you can see, the M2 motor cable only just reached its socket on mCore.

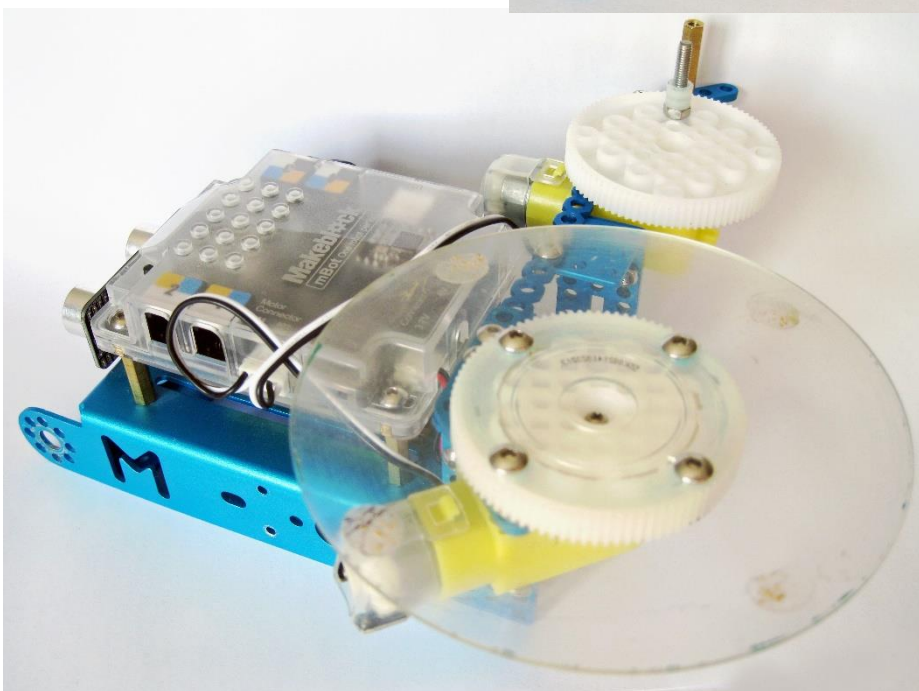
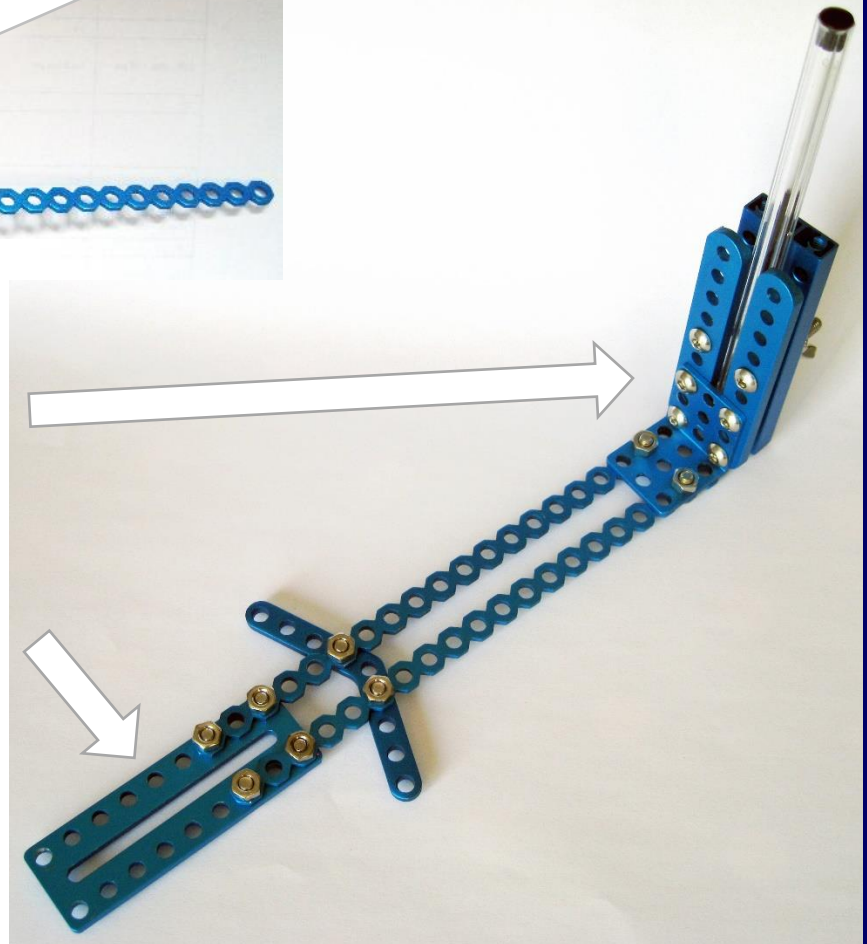
Holding the pen firmly in these drawing machines is also a problem. The '*KidsandChips*' model and many others on the internet are rather wobbly, just relying on rubber bands to hold a pen against an upright.



I rather liked my first attempt, the 'claw-ended grab' shown on the left, but I didn't think that this was firm enough and spent ages trying to invent something better.

I was rather pleased with the one that I finally settled on using. This is shown on the right (at the far end of the pivot arm) - which for the crank to work correctly needs the pivot point to move in a slot.

You can see clearly how I added a slotted plate for this and the 135° angled plate that I used as the crank-pivot point. The picture below shows the two drive wheels added to the motor support arm. The one in the foreground has the CD bolted to it - its cardboard top has been moved for clarity.



The rear drive wheel has a 'crank-pin' passing through it. This is an M4 x 30mm screw held in place with a nut and then a 10mm plastic spacer added. When the arm is finally added on top of this, then it is held in place with an M4 'Nylock' nut.

The M4 x 25 screw used to pass through the slot in the arm and down into the brass pillar to provide the pivot point for the arm has a 2.5mm plastic spacer on either side of the slot.



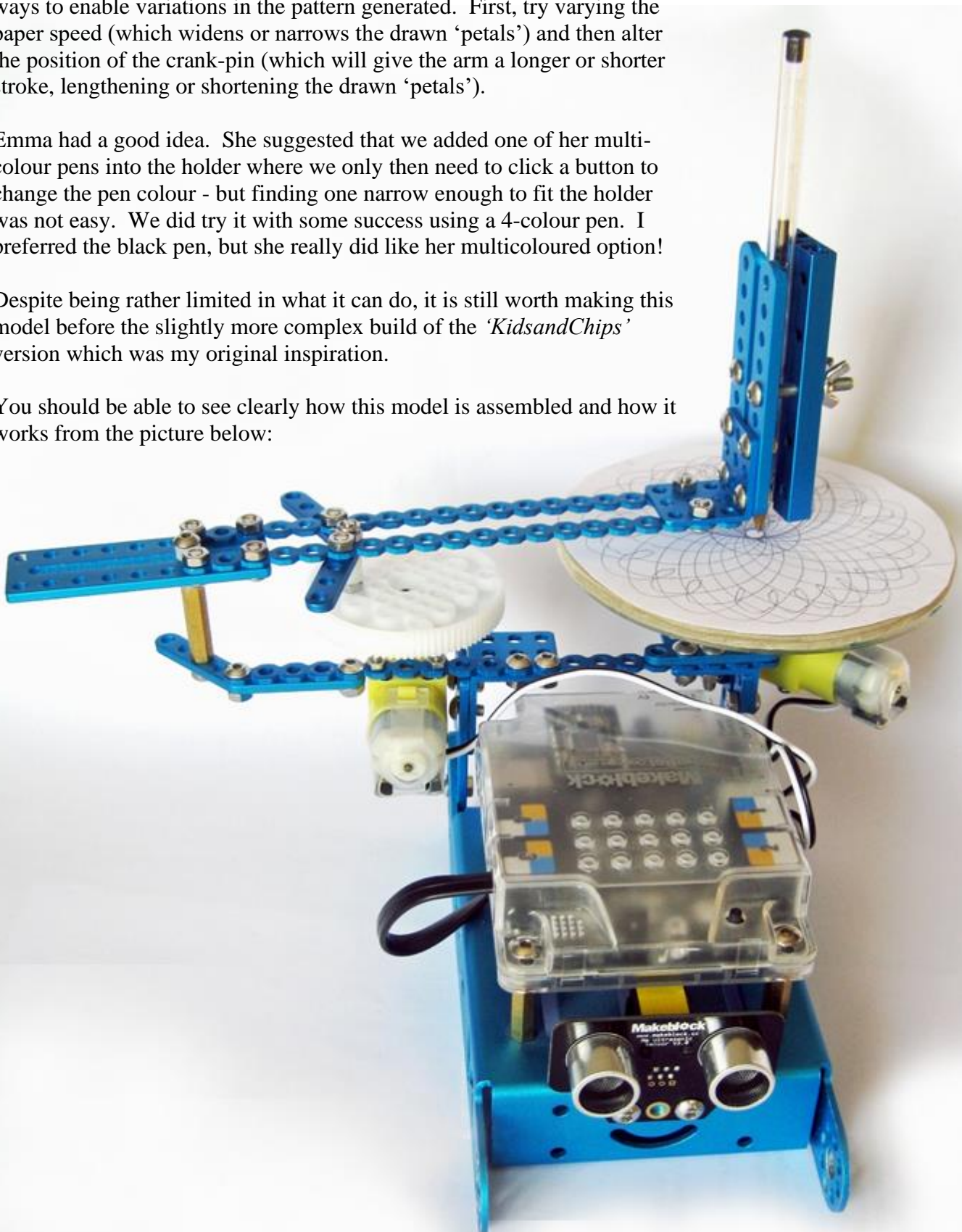
mBot and Me *a Beginner's Guide*

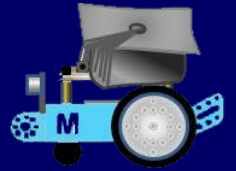
With this very simple 'roulette' drawing machine, there are only two ways to enable variations in the pattern generated. First, try varying the paper speed (which widens or narrows the drawn 'petals') and then alter the position of the crank-pin (which will give the arm a longer or shorter stroke, lengthening or shortening the drawn 'petals').

Emma had a good idea. She suggested that we added one of her multi-colour pens into the holder where we only then need to click a button to change the pen colour - but finding one narrow enough to fit the holder was not easy. We did try it with some success using a 4-colour pen. I preferred the black pen, but she really did like her multicoloured option!

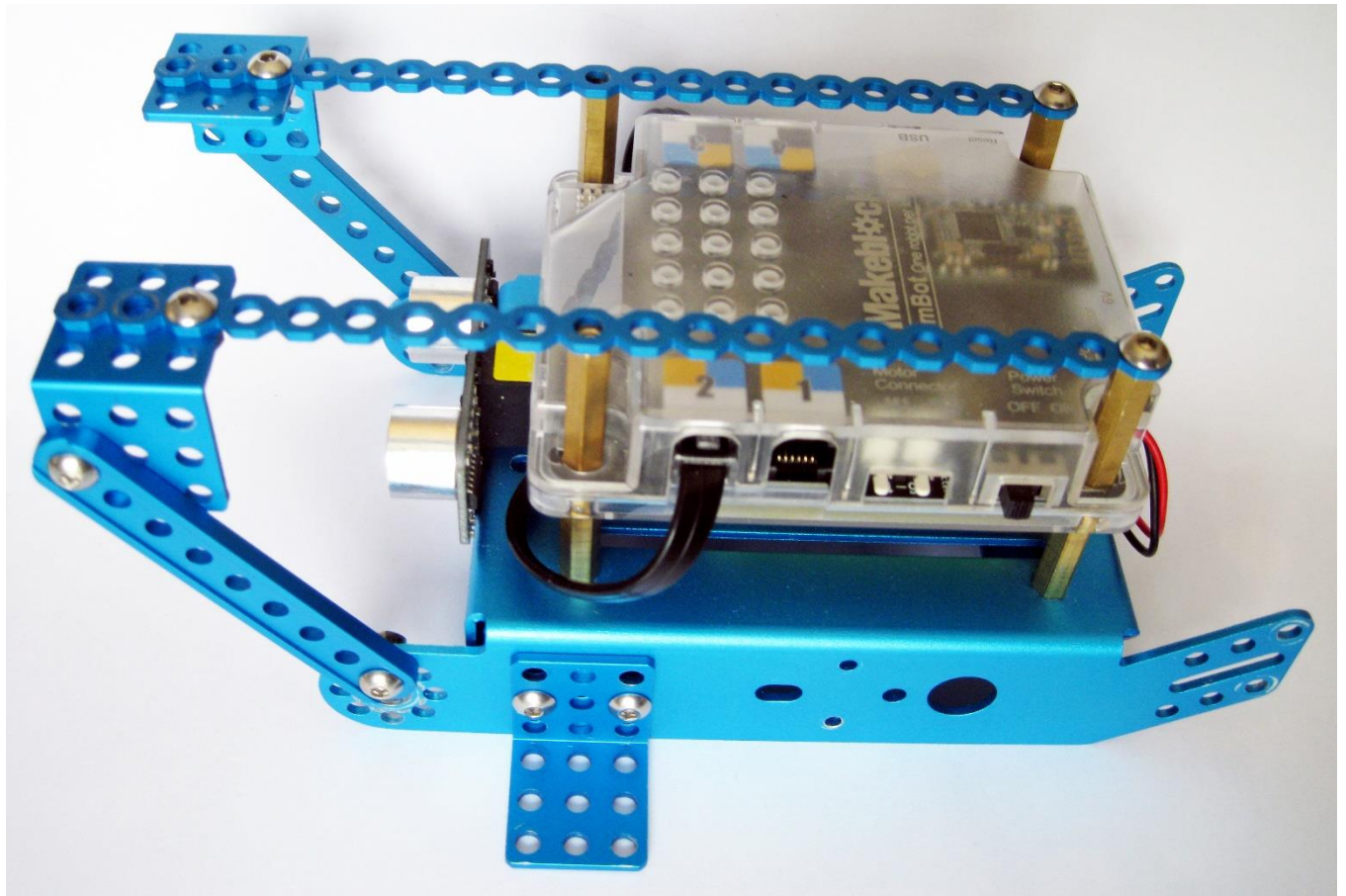
Despite being rather limited in what it can do, it is still worth making this model before the slightly more complex build of the '*KidsandChips*' version which was my original inspiration.

You should be able to see clearly how this model is assembled and how it works from the picture below:





To emulate the 'KidsandChips' version (which, with its second arm, can produce more pattern varieties) required a rather more complex build. I started by removing the four screws that held mBot's plastic cover in place over the mCore circuit board and then re-secured the cover by adding a second set of four 25mm brass spacer pillars with their screw ends passing through the cover and into the threaded holes in the brass pillars below. I used the largest hexagonal hole in the body of the little spanner to tighten these - firmly, but not too tightly since they were compressing the plastic casing.



I then used two M4 x 8 screws to attach the end hole of a complete (160mm) length of 20-hole cuttable Linkage to two brass pillars at the rear of mBot. Do not add any screws - *yet* into the front two pillars. This will be done when you add the motor beam shortly. To the outer ends of each piece of linkage, I added a 3-hole x 3-hole right-angle bracket using just one 8mm x 4mm Screw and Nut as shown above.

To give this construction some rigidity, you can also see clearly in this image that I added a 9-hole, 76mm 0412 beam to act as a bracing strut using two 8mm x 4mm Screws and Nuts (the bottom one passing through the topmost hole of those that form a circular pattern at the front of mBot). I did the same on the opposite side of mBot too. You will also notice that I added a further 3-hole x 3-hole right-angle bracket to the just the left-hand side of mBot to provide a little extra width to the chassis base, making a supporting 'foot' (fitting flush with mBots chassis base).

I used two 8mm x 4mm Screws and Nuts passing through the 'M' shape cut into mBots left side to secure this in place. I did not need to do this on the opposite side since, as you will see on the next page, the left side of the motor beam assembly overhangs a little more than it does on the right and the one remaining right-angle bracket at my disposal was required elsewhere in the model.

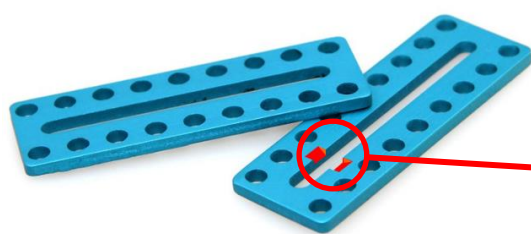


mBot and Me a Beginner's Guide

The motor beam is constructed from three 9-hole x 2 (Code 11) Slotted Plates with the motor for the paper rotation wheel mounted below it (to get it the wheel as low as possible) and the motor for the drive crank mounted above it to get the wheel as high as possible so that the crank linkages (and pen carrier) would pass easily above the rotating paper.

When I started constructing this, I found that when mounting the paper rotation motor underneath one of the 9-hole x 2 (Code 11) Slotted Plates, it didn't quite sit flat under the beam. This was because the both sides of the motor have a clip which is raised slightly (about 1mm) higher than the body of the motor (see the diagram on the right).

N.B. These two clips secure a plastic cover (end-cap) which protect the motor terminals. So, decision time, file down the clip or what ... ?



I decided that the simplest solution was to modify one of the slotted I1 plates (shown on the left) slightly by using a needle file to cut a small chamfer in the inner slot of the plate (between the second and third holes in from one end - as shown here) - just a small adjustment like this, on just one plate, allows the motor to sit flat. This was very easy and has not prevented the plate from being used in any other

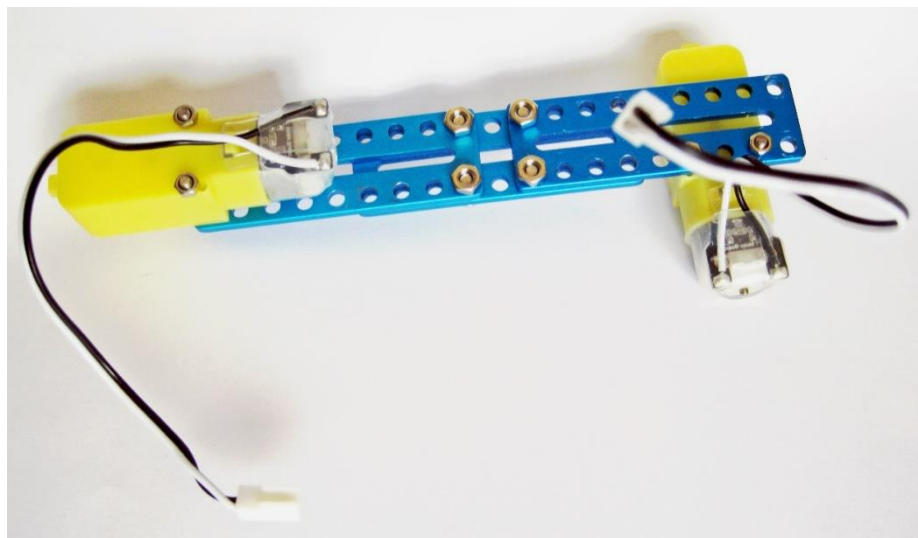
way. The paper rotation motor is attached to the modified I1 plate with its spindle in line with the plate (using the end two holes) and using the same two bolts that were used to secure it to mBot's chassis.

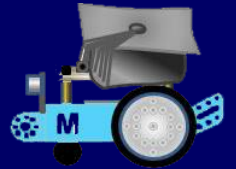
The motor for the drive crank is mounted at right angles across the second I1 plate using the second and fourth holes on the back-edge of the plate; and once again, using the same two bolts that were used to secure it to mBot's chassis.

The axles on each of the motors should both point upwards. The two plates with their motors attached are then joined together using the third 9-hole x 2 (Code 11) Slotted Plate using just four 8mm x 4mm Screws and Nuts.

The picture on the right shows the motor beam fully assembled - but is shown from the underside. Do note the

positioning of the long bolts passing through each of the I1 slotted plates and more importantly note the separation of the two motor plates by the central holes in a third (central) slotted plate. You should also be able to see that the paper rotation motor has its weight much further out to the left of centre - hence the addition of the aforementioned supporting 'foot' to the left side of the chassis base.

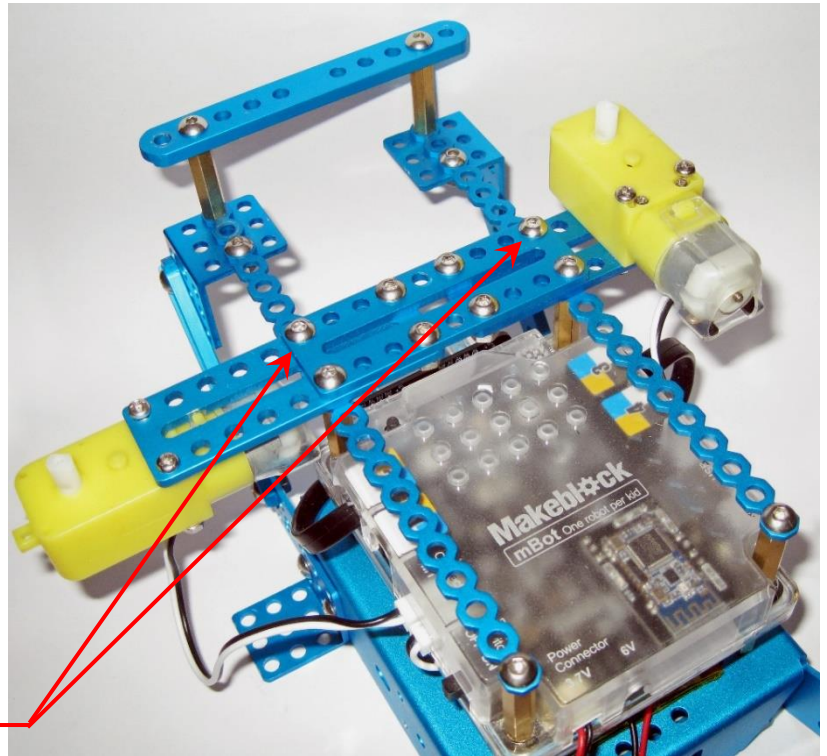




I positioned the motor beam centrally on top of the completed chassis base unit and used two 14mm x M4 Screws passing through the two overlapping thicknesses of the I1 plates and the longitudinal cut linkages. The screws go through the holes in the rear of the motor beam assembly and can be fully tightened into the brass pillars below.

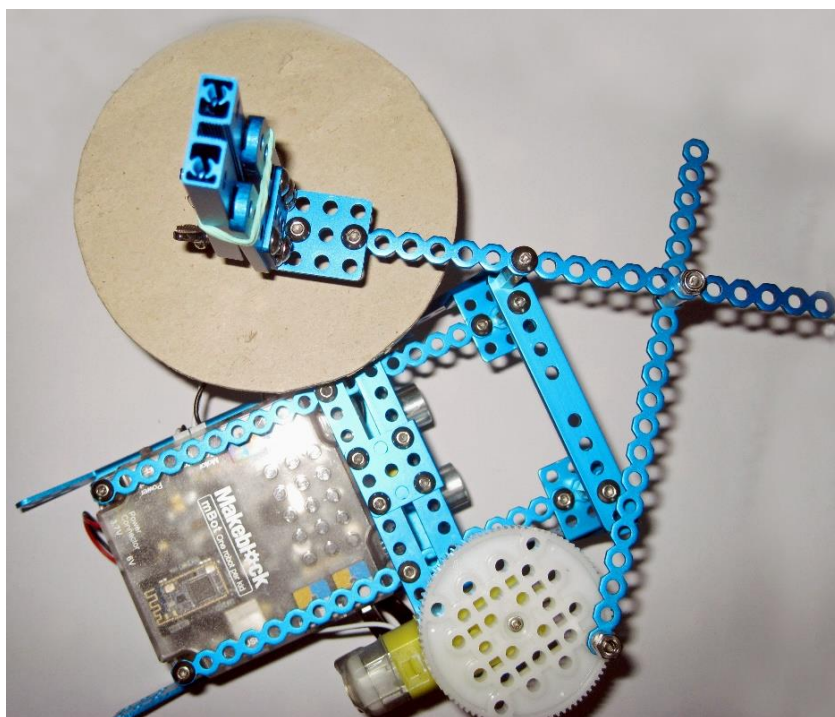
This created a solid assembly and although not strictly needed, I added two extra 14mm x M4 screws through the front two holes in the two overlapping thicknesses of the I1 plates and the longitudinal linkages, securing them with M4 nuts underneath.

This looked much neater as you can see in the image on the right.



You can also see in this image that to complete the support system needed for the pen carrier and crank linkage pivots, I added two more 25mm hexagonal brass pillars to the front ends of the longitudinal cut linkages securing them underneath the angle brackets with two M4 nuts and on top of these pillars I added a 10-hole, 90mm 0412 beam.

Next, I fitted the two drive wheels back on to the axles on each of the motors, securing them with the small self-tapping screws carefully saved when mBot was disassembled earlier. I then connected each of the motors to the mCore board using socket M1 for the paper rotation motor and socket M2 for the linkage crank motor.



I used my paper support plate (CD / cardboard assembly) from the last model and used the same pen holder from that model too. You should be able to work out from the picture on the left and the one shown at the top of the next page how the arm linkages work.

Do aim to keep all of the arm linkages as *horizontal* as possible.

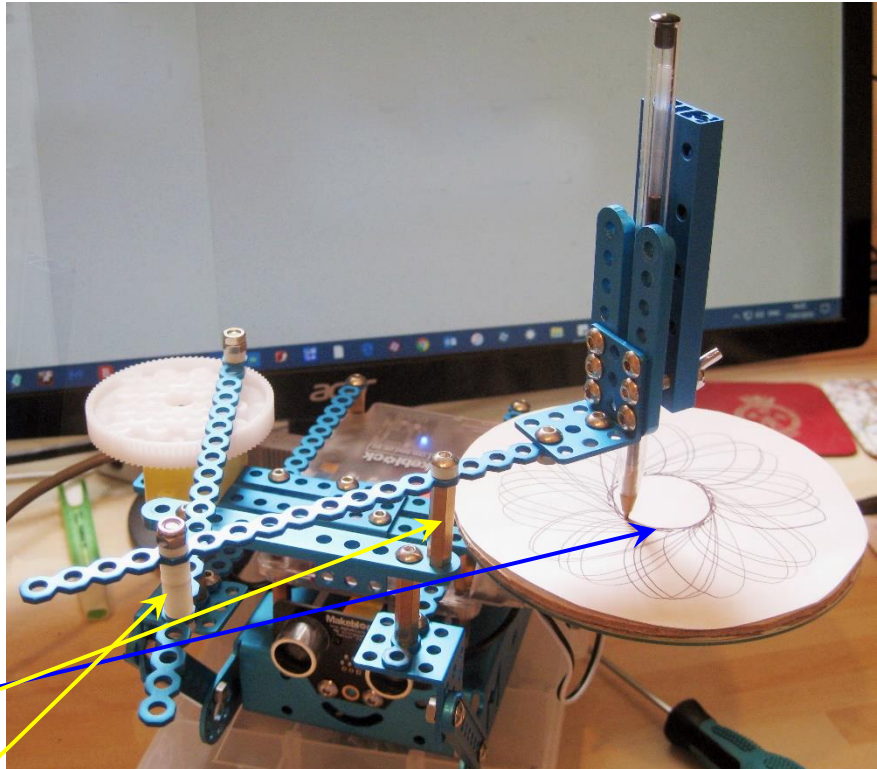
N.B. The primary pivot post needs more height to position the pen arm correctly - use a 20mm x M4 screw with 2mm spacer on top of the pen arm and 10mm spacer underneath it.



The M4 threaded hole in the top of the brass pillars is 9mm deep and works well here. With the primary pivot positioned in the eleventh space (from the outer end of the pen arm) puts the pen, more-or-less, over the central axis of the paper rotation motor – you might get a small circle created here if it is not perfectly central.

Moving the primary pivot position into the 12th, 13th or 14th hole positions on the pen-holder arm causes a fully drawn pattern to create a circle created by the overlapping lines with a radius equal to the pitch of the holes in the linkages (8mm).

The secondary pivot on top of the crank arm uses a 14mm x M4 screw with a 2mm spacer on top passing through the pen arm and screwed firmly tight with an M4 nut underneath it. Fifth space (from the outer end) on the pen arm and sixth space (from the outer end) in on the crank arm seems to be a good place to start.

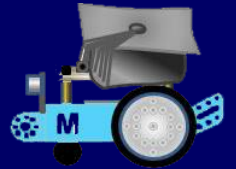


All of the 25mm brass pillars have an M4 threaded hole in the top & these holes are 9mm deep. Use a 14mm M4 screw in the top of the secondary pivot and a 22mm M4 screw in the top of the primary pivot; these should stop firmly at the bottom of the hole (providing a bit of friction to prevent them from unscrewing) and leaving just enough movement for both of the linkages to turn freely. Use an M4 Nylock nut on the underside of the secondary pivot to fix it firmly to the crank linkage; since this needs to be repositioned to create different patterns, I used an M4 wing nut to do this.

For the crank fitted to the outer ring of holes in the crank-wheel, I attached the linkage using an M4 x 22mm screw with a 2mm spacer on both sides of the linkage and an M4 Nylock nut on the top. Check the screws on top of the two pivot pillars are tight - the primary pivot of the pen arm, especially, can work loose.

All drawn output from this model will still produce variants on the 'petal' theme. As before, increasing the paper motor speed creates a wider spacing of the lines whilst decreasing it produces a finer spacing. Repositioning the pen arm on the top of the primary pivot moves the pen out further from the central axis giving a larger diameter central 'hole' where the created patterns create a circle at the centre of the 'petal'.

Finally I tried a totally different type of drawing machine to the two rotating paper 'roulette' types outlined above. ***This turned out to be very much simpler to build, to experiment with and so much more satisfying to watch in use.*** In this model variant, the paper remains fixed whilst mBot's two motors drive a linkage system via two rotating cranks to push-and-pull a pen across the paper.



This type of model is generally referred to as a '*Pintograph*' and there are many examples of these to be found on the internet. I found a particularly useful website created by American, Wayne Schmidt where one of his pages explains the practicalities of pintograph construction and use in some considerable depth. This can be found at:

<http://www.waynesthisandthat.com/pintographs.html>

'*Pintographs*' are really *harmonographs* that use electric motors instead of pendulums to move a pen to create detailed line drawings (Lissajous curves). Varying the motor drive systems and arm linkages will create unique images. It's possible to have several sets of arm lengths for a pintograph machine so they can be changed over to create different designs; and by carefully selecting the start & stop points or stopping a drawing part-way through its drawing cycle an even wider variety of designs can be created.

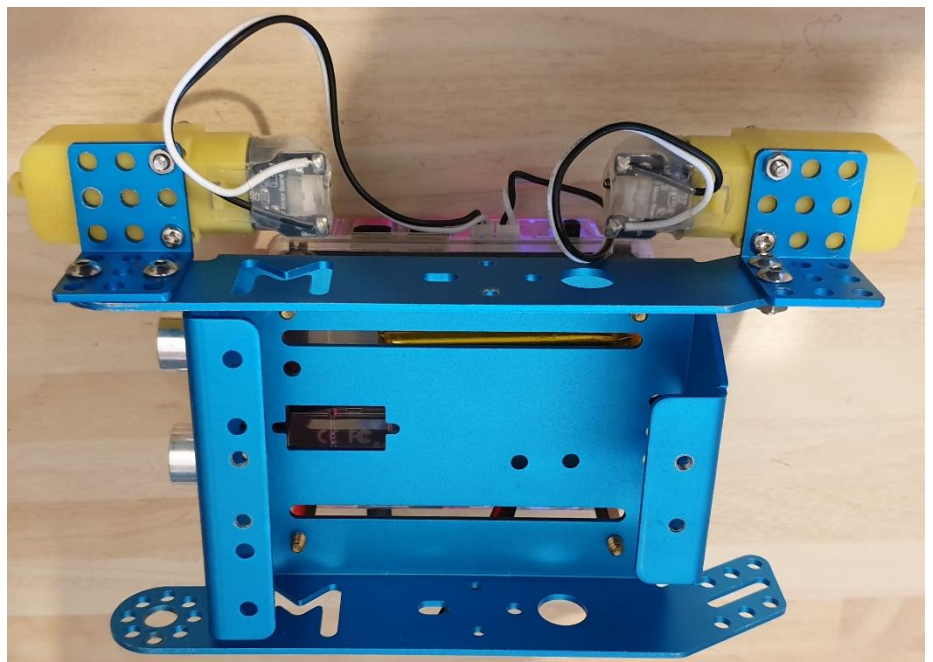
'*Pintographs*' come in two basic forms, either the '*Straight Arm*' or the '*Scissor Arm*' type, each producing similar images. The main difference between them being that the '*Scissor Arm*' amplifies pen movements to produce larger drawings in the same way as *pantographs* enlarge drawings. More importantly, they also create designs radically different from anything produced by '*Straight Arm*' types. Both model variants are shown in illustrations on the next two pages.

The relative motion of the cranks (*which must rotate at slightly different speeds*) generate harmonic relationships, with the pen being continually being moved along different paths bounded by four extreme points. With Pintographs, the faster motor will eventually catch up with the slower one; so drawn figures will eventually repeat themselves if the machine is left to run for long enough. Ideally **one motor should rotate one-tenth of a revolution faster than the other**. Otherwise, if a pintograph has both motors running at exactly the same speed, then they will push-and-pull the pen repeatedly over the same path creating a (more-or-less) single solid line!

It's critical therefore to balance the speeds of the two motors. If the speed difference between them is too great, then the lines will be so far apart that lines won't appear to flow smoothly; but if the motor speeds are too close then the lines will be inclined to overlap as a solid mass of ink.

I found it comparatively easy to mount mBot's two geared motors onto either end of the basic chassis using two 3-hole x 3-hole right-angle brackets as shown on the right. Once you have fitted the two drive wheels back on to the motor spindles (and located them with their little self-tapping screws, then that's it!

This is all you need to do to create a pintograph drive-unit to which you can add two or four arms made from standard (160mm) lengths of 20-hole cuttable Linkage strips.





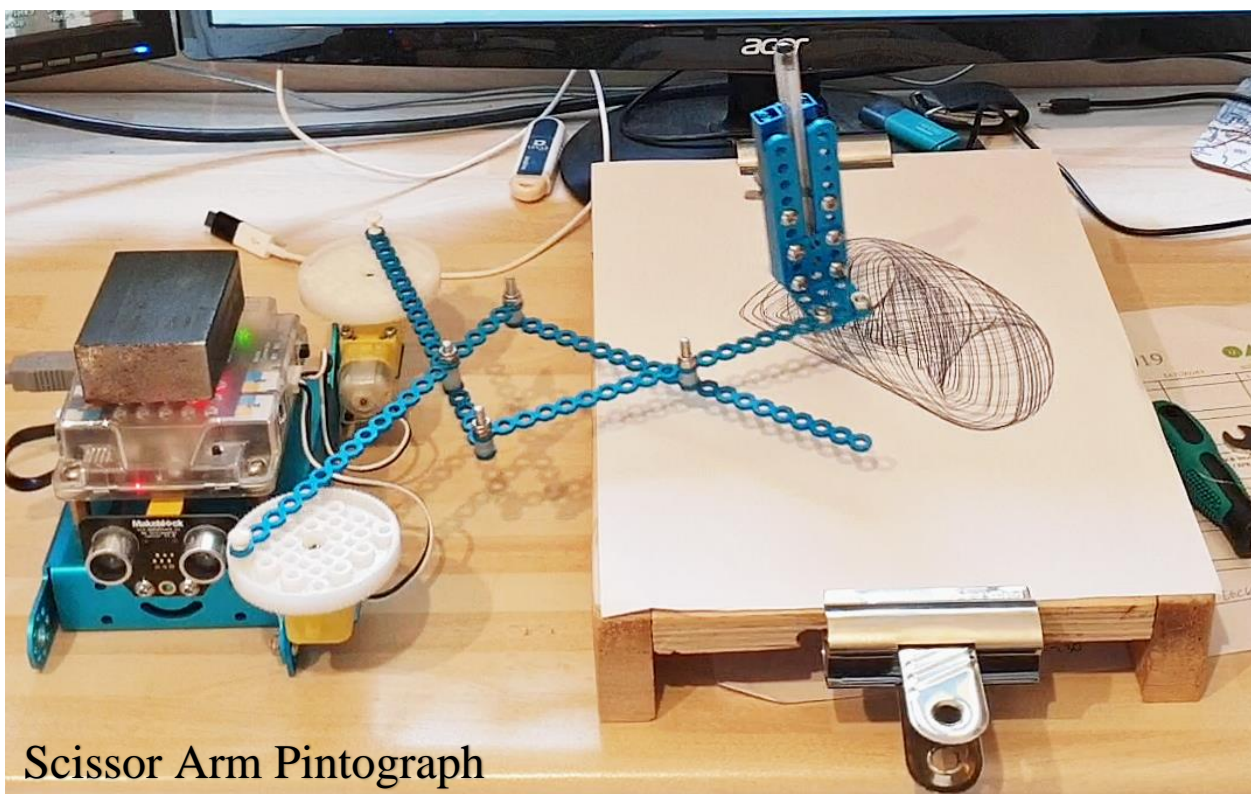
I decided to continue using my fairly successful pen holder clamp (shown in the illustrations of the previous two models). You can experiment with you own pen holder ideas here very easily, just as you can with the many variants of two-arm or four-arm (scissor) pivot points and crank positions. It is worth noting here that all of your arm assemblies are a separate entity to the pintograph drive-unit described at the bottom of the previous page - you just need to connect the two together - a suitable crank pin attaching the drawing arms to the two drive wheels.

I did experiment with using screws (with plastic spacer washers and lock nuts) as the crank pins for my early models but they were so tedious to undo each time that you wanted to separate the arms from the drive-unit or to change crank pin positions that I substituted them for two of the longest (15mm) Plastic Peg Rivets that I had. These worked surprisingly well, enabling me to extract them quickly and they stayed in position (mostly) whilst drawing was in progress.

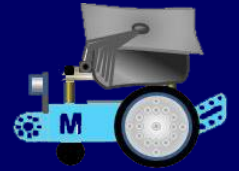
I also tried using Plastic Peg Rivets in lieu of screws and nuts to make quick-change pivot points for the pintograph arms, but these became loose far too often; so (as you can see below) using screws with plastic spacers and lock nuts on the arms is the only real option for model reliability.

The final problem to solve, was the height of the drive cranks. Because of the positioning of the motors, the top of both drive wheels is approx. 60mm above the base of mBot's chassis. I tried extending a pen down through my pen holder clamp this far and whilst it did work, its leverage on the arms of the Pintograph was excessive. The only solution was to make a purpose-built paper holding platform about 50mm high and the same dimensions as a piece of A4 paper (see the illustration below). A solid piece of thick (18mm) plywood cut to match provided the top, this was supported on two strips of timber to gain the height required. I then added two 'Bulldog' spring clips to hold the paper in place.

This worked very well and had the great bonus of firmly holding the paper steady whilst drawing.



Scissor Arm Pintograph



Remember, the drive unit is a separate entity used to pull and push all variants of Pintograph arms and for some arm combinations, the drive-unit would (thanks to the overall weight of the chassis assembly) remain unmoved - *but* - for some combinations, the push/pull action of the arms would move the drive unit about a bit. I could have connected mBot's chassis (the drive-unit) to my wooden paper holder somehow but this would have made the whole thing rather cumbersome and unwieldy to move.

I resorted instead (as you can see in the illustration at the bottom of the previous page) to adding a weight to the top of mBot in the form of a block of steel (50 x 25 x 75) that I just happened to have. This worked very well and stopped the drive unit from being moved by the action of the drawing arms.

For all of my drawing projects I used the two very simple scripts shown on the right to start and stop the motors. Note that for this model, one motor is set (as mentioned earlier) to be 1/10th faster than the other.

```

when up arrow key pressed
  DC motor motor port1 clockwise rotates at power 30 %
  DC motor motor port2 anticlockwise rotates at power 33 %
  
```

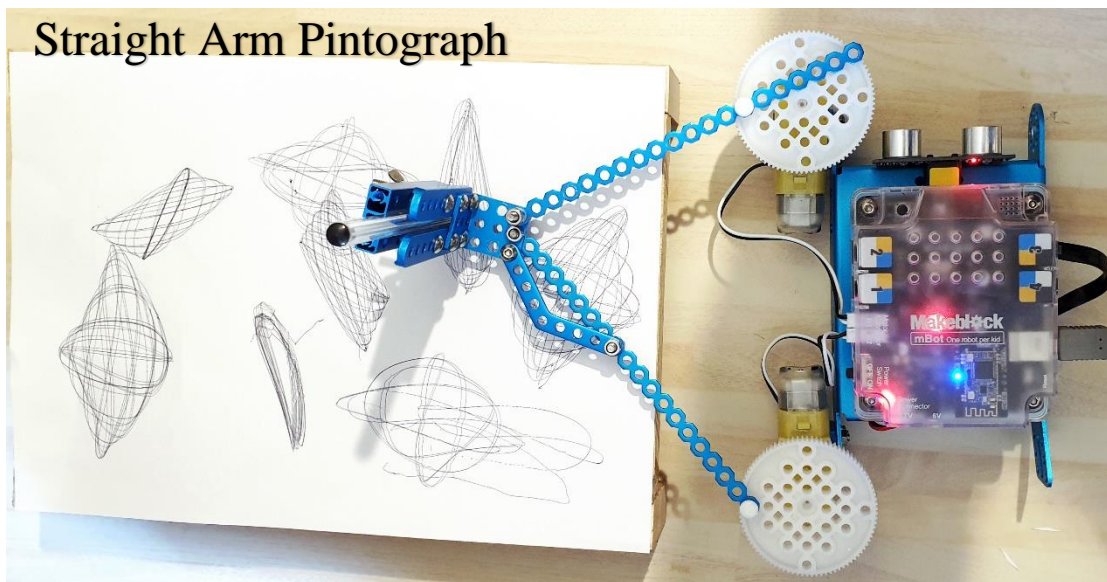
Using keyboard key-presses to start *and more importantly*, **stop** the motors in a hurry was important too.

```

when down arrow key pressed
  stop moving
  
```

Running mBot's TT geared motors at 25% is the lowest you can go without them stalling; but under the extra load from the pintograph arms (dragging a pen), my recommendation is to set one motor to 30% which is about 2 secs per revolution (speed 76/255) and the second motor to 33% (84/255).

Straight Arm Pintograph



Changing the rotational direction of the motors (or preferably just one motor), the crank radius, the arm length, the clamp pen position and pen weight pressure all affect the final drawing.

N.B. If there isn't enough

weight on the pen, then it may lose contact with the paper, creating areas of the drawing which are too lightly drawn, or even missing altogether. The position of the pen relative to the pivot point at the outer end of the arms has an enormous influence on the drawing too. Many pintographs have their pens slightly offset to one side of the pivot and pens can also be positioned inside or outside the pivot point.

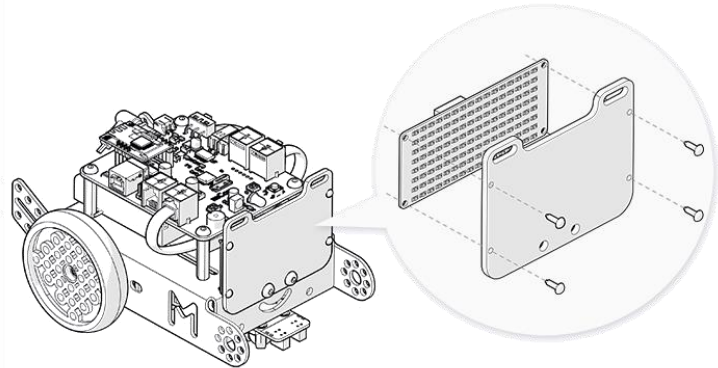
Reducing the crank radius compresses the image whilst increasing the arm length stretches the image left and right and compresses it top to bottom. ***Do experiment with all of these model concepts.***



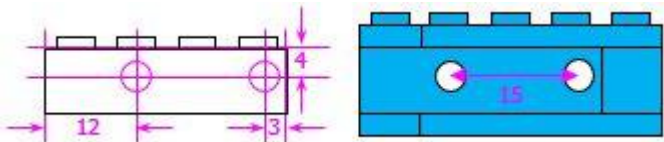
Appendix 1 - mBot 'add-on' component - LED Panel

MakeBlock Model No. 13412 - the LED matrix display board

'LED Panel' is the name now given to Makeblock's 8 x 16 LED Matrix display board. It was initially designed to show simple bitmap graphics to provide a 'face' for mBot (to be mounted above the little mouth shape cut into the front of mBot's chassis as shown on the right). Previously therefore, it was known as just 'Face'. Matrix is word that describes a rectangular array of anything arranged in rows and columns. This is a good first purchase 'add-on' and very many mBot users really do recommend it. It cost me about £10 + £3.50 UK postage.



Begin by removing the protective paper from the diffuser plate and attach the plate to the LED matrix board with four little plastic pegs (as shown above right) and then connect the LED panel to port 1 of mCore. The LED matrix assembly can then be attached with two M4 bolts to the robot as shown. The mounting method shown in the illustration above is fine if you just want to give mBot facial expressions (or 'emotions'), but you must disconnect the ultrasonic sensor to fit it here, which is not a good option in my opinion!



My own solution was to drill two 4mm holes @ 15mm centres through a spare 4 x 2 stud Lego building-block filched from Emma's toy box. See the diagrams on the left.

A single Lego block with the LED matrix display bolted to it fitted neatly on to the studs moulded into the top of mBot's cover (but why are they in a 5 x 3 configuration? - possibly because the two centre studs are on the centre-line of mBot I guess? Whereas all Lego stud plates are in a 'multiples-of-two' matrix).

A single asymmetric block looked a bit odd; and this was fine on its own, but by adding several more purloined bricks to build the small wall shown in the diagram above it became a much stronger mounting beam that neatly spanned the five studs on mBot. Emma can always get the Lego blocks back - eventually!



See the illustration above right. The LED Panel is quickly and easily removed for storage too (and for disassembly I fitted the two M4 x 25mm bolts with M4 wing nuts). The Lego mounted LED Matrix display can be easily fitted facing forwards, backwards or sideways on the studs on mBots cover and is very useful when displaying feedback from the sensors. I use this a lot, but I've not seen any web or other reference to anyone else mounting mBot components in this way.

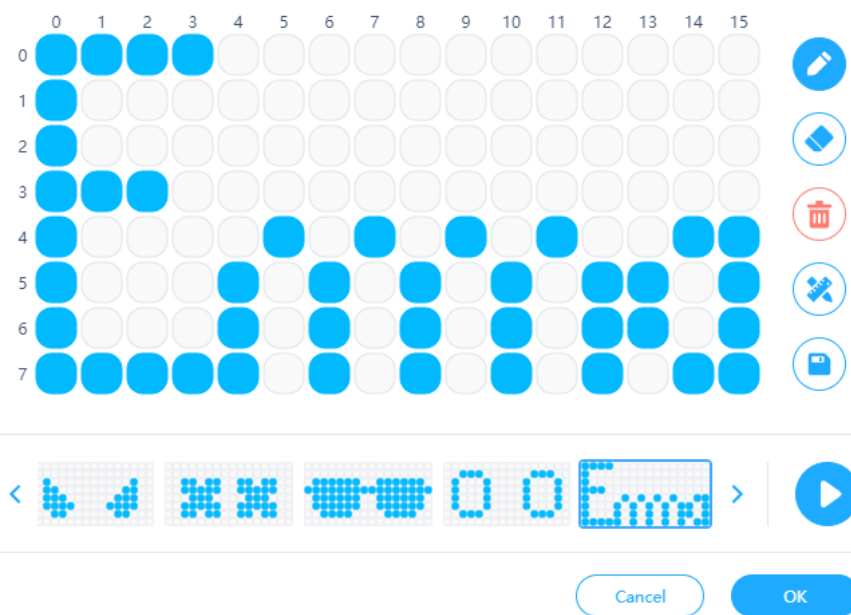


MakeBlock's LED matrix display board is described as 8×16 (read 'eight by sixteen'), because there are eight rows and sixteen columns of individual little Light Emitting Diode lights which can be turned on and off using the 'shows image', 'shows text', 'shows number' and 'shows time' stack blocks.

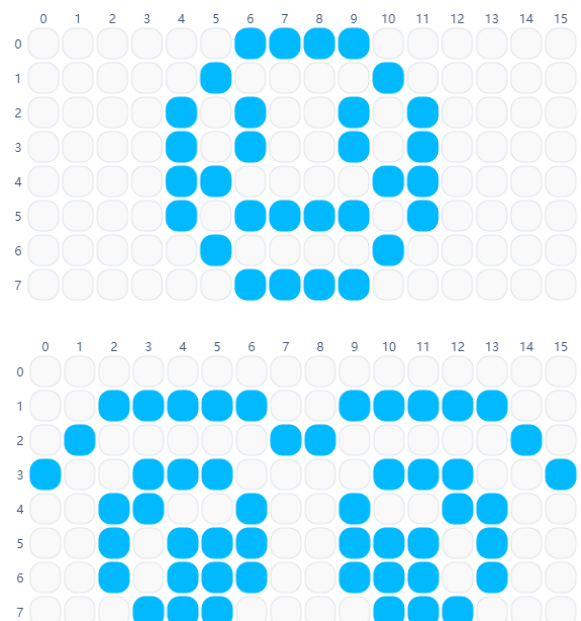
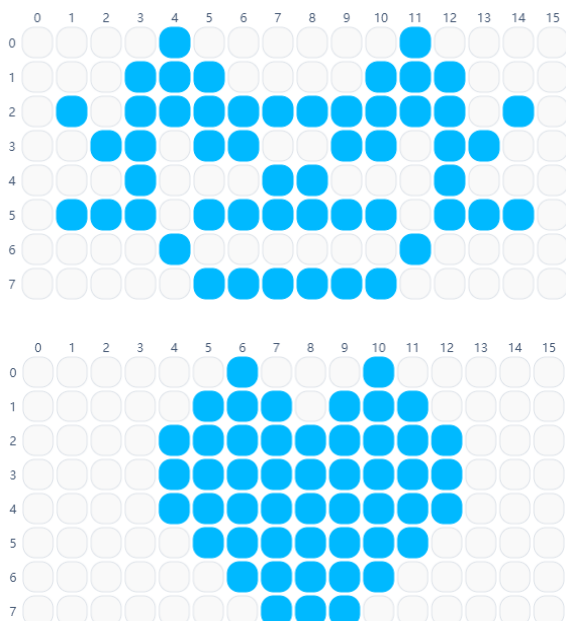


Remember, you need to be on the 'Devices' tab to use these robotics blocks. If you drag any of the 'show image' stack blocks from the 'Show' blocks category on to the 'Scripts' Area and then click on the little box with the two blue oval 'eyes' showing then it opens-up the edit 'Panel' (see below):

This is much improved from previous versions. There are ten samples here to choose from but it's fairly intuitive to use and you just click on the blue circles to light-up or clear a representation of each of the individual diodes of the 'Panel' display to create your own image; it's rather like a huge pixel painting screen. You just press the 'floppy disk' icon to save your creation and add it to the list.



Emma loved doodling on this using my Surface Tablet's touch screen and was (as you can see left and below) instantly at home creating pictures. MakeBlock describes these 'Panel' images as 'emotions'.





mBot and Me a Beginner's Guide

Several of the pre-defined 'emotions' are 'eye' graphics (just like the ones shown in the simple script on the right). This script illuminates the LED matrix with each chosen 'emotion' and then uses a 'wait' block (set to 1/10th second) after each in turn to create a simple animation and then turns the display off at the end of the sequence.

Try making this and the variant shown below. It too shows the same four simple 'eye' graphics, but this time it uses a different stack block which includes a timer (also set to 1/10th second).

```

when clicked
repeat 10
  LED panel port1 shows image [eye graphic] for 0.1 secs
  LED panel port1 shows image [eye graphic] for 0.1 secs
  LED panel port1 shows image [eye graphic] for 0.1 secs
  LED panel port1 shows image [eye graphic] for 0.1 secs
LED panel port1 clears screen

```

```

when clicked
LED panel port1 shows image [eye graphic]
wait 0.1 seconds
LED panel port1 shows image [eye graphic]
wait 0.1 seconds
LED panel port1 shows image [eye graphic]
wait 0.1 seconds
LED panel port1 shows image [eye graphic]
wait 0.1 seconds
LED panel port1 clears screen

```

The four blocks are all inside a 'repeat' block which is set to repeat the animation ten times and in doing so, it creates animated 'blinking' eyes. This is a much neater solution to the first script, so learn from this and *choose from block categories wisely!*

Shorter scripts are more efficient.

You can create many animation sequences like this with your own graphic creations.

There are also four rather useful 'shows nnn' stack blocks in the 'Show' blocks list (three of these are shown on the right) - the first is for displaying numbers - up to four digits (feedback from sensors etc.). The last block will display a time value (set by script values perhaps?).

```
LED panel port1 shows number 9999
```

```
LED panel port1 shows text CAT at x: 0 y: 0
```

```
LED panel port1 shows time 12 : 59
```

There are two that can be used to display user or script generated text (upper or lower case BUT only three characters at any one time. You can try experimenting with the middle of the three blocks shown above, changing the values in the x: bubble window. Try giving x: the value '2'. You will find that this puts a two-character message e.g. 'Go' in the centre of the LED matrix panel.

You can use these 'offset value' settings to usefully 'scroll' much longer pieces of text across the LED panel using an (x = x - 1) repeating loop. An example of this is shown at the top of the next page. You will see that it moves the text (rather slowly) to display any message longer than three characters. It needs a very short (0.01 sec) 'wait' block timer inserted into the script to allow the LED Panel time to refresh itself - and it will not work without this short pause! You can also try something similar (see the next page) by using a (y = y + 1) repeating loop to 'scroll' text downwards.



Start by making two simple 'Variables' called 'x' and 'y' and then build and test the script shown on the right substituting your own text characters to suit (sadly, this seems to be limited to 19 chars. max.):

N.B. This script will stop after 19 characters have scrolled past, but you may need to click the 'stop' button (or the script 'hat' block) to halt the script if the text is very much shorter.

The script shown below is a variant of the one above right and will scroll the text 'Go' vertically downwards (starting above the LED panel and exiting below it).

```

when clicked
  set x to 0
  set y to 0
  forever
    LED panel port1 shows text Hello Emma at x: x y: y
    change x by -1
    wait 0.01 seconds
    if x = -120 then
      LED panel port1 clears screen
    stop all
  
```

```

when clicked
  set x to 2
  set y to -7
  forever
    LED panel port1 shows text Go at x: x y: y
    change y by 1
    wait 0.01 seconds
    if y = 9 then
      LED panel port1 clears screen
    stop all
  
```

Downward scrolling is a good demo. - but not actually very useful! Do note how the 'If' block loop in both scripts exits the 'Forever' block loop.

The 'shows number' stack block is very useful. It performs rather better than the 'shows text' stack block since it can display up to four numbers on the LED matrix panel.

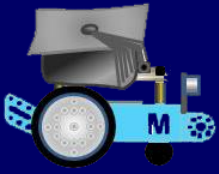
This block is therefore a good way to displaying numeric feedback from sensors rather than just having feedback displayed as a Variable on mBlock's 'Stage'.

It's worth noting here that you can position numbers on the LED matrix (just as if they are text characters).

If you insert numbers into the 'shows text' stack block then you can offset the position of those numbers by inserting a suitable value into the (x:) window; but you can only show up to three characters doing this - but useful perhaps if you want to centrally position just one or two digits.

Do remember though that the generally much more useful 'shows number' stack block displays up to four numbers.

The 8 x 16 LED Matrix display 'Panel' can be a very useful output device - that is until you can get your hands on the rather desirable *Me 7-Segment Display module* which will do a similar job so much better!

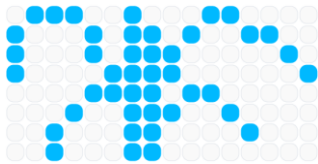


mBot and Me a Beginner's Guide

To demonstrate the number-offset trick using the 'shows text' stack block mentioned on the previous page you first need to make a Variable called 'Counter' and then create the script shown on the right.

This script uses '5' in the repeat loop as an 'offset x value' to position a single digit in the centre of the LED Matrix display; but first the script checks to see if the counter has two digits and if it does then it sets the 'offset x value' to '2'.

We first wrote this using the mBlock 3 version of the software and Emma was very taken with this countdown timer routine. She insisted that we added a sound file named 'bang' at the end. In her version, she also added a 'shows image' stack block with the 'Firework' display image shown below:



I didn't think much to it, but she thought that it was brilliant and created loads more images to try. The LED Matrix display 'Panel' really got Emma into thinking about scripting for herself - **what a result!**

```
when clicked
  set Counter to 10
  if Counter = 10 then
    LED panel port1 shows text Counter at x: 2 y: 0
    wait 1 seconds
    change Counter by -1
  repeat 10
    LED panel port1 shows text Counter at x: 5 y: 0
    wait 1 seconds
    change Counter by -1
  LED panel port1 clears screen
  set Counter to 0
```

It was not possible to add this sound file into the mBlock 5 script shown above (because sounds are only possible when they are called by a 'Sprites' tab script) - and here we are on the 'Devices' tab!

```
when I receive Play_Bang
  play sound bang until done
```

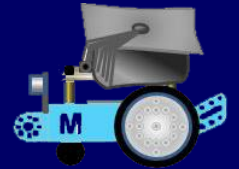
But by now, we know all about 'Broadcast' messages (see Chapter12 - page 76) so this is easily fixed ...

```
LED panel port1 shows image [Firework]
broadcast Play_Bang
wait 2 seconds
```

... I switched to the 'Sprites' tab and uploaded the same sound file into my 'Sound Library' and created the two-block script shown above left. Next I returned to the 'Devices' tab and added the three blocks shown above right to the main countdown script listed at the top of the page.

I positioned these three blocks below the 'repeat' loop and above the 'clears screen' stack block. (The 'wait' 2 seconds block has been added here to match the length of the sound file, so that Emma's 'Firework' image is 'extinguished' as soon as the 'bang' sound file ends).

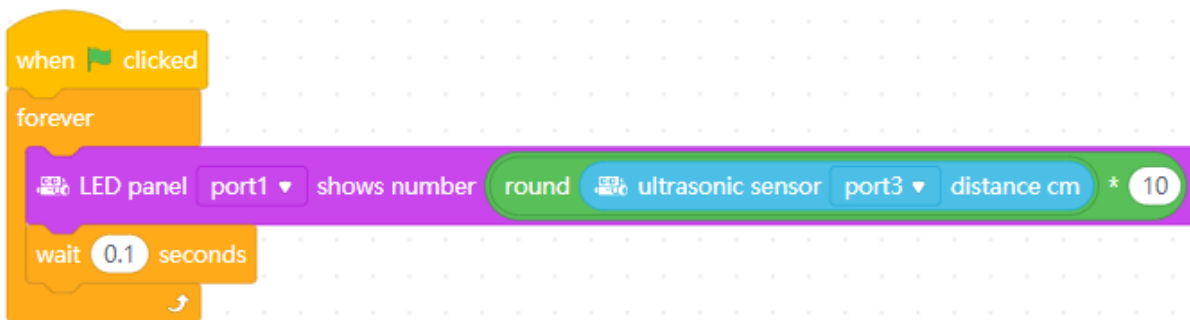
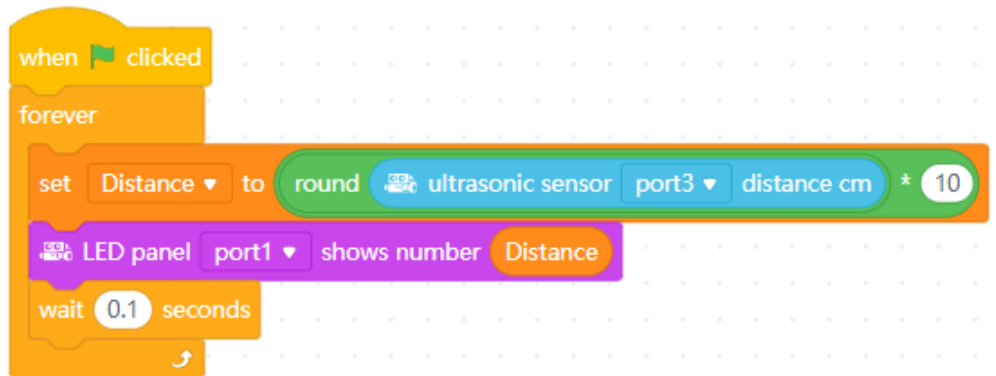
The script shown at the top of the next page uses a Variable named 'Distance' to take the reading from the ultrasonic sensor and convert it into millimetres. It then displays (in real-time) the reading as continuous feedback on both mBlock's 'Stage' and on the LED Matrix 'Panel'.



You do not have exhibit a Variable on the stage (or even use a Variable at all) if you don't want to.

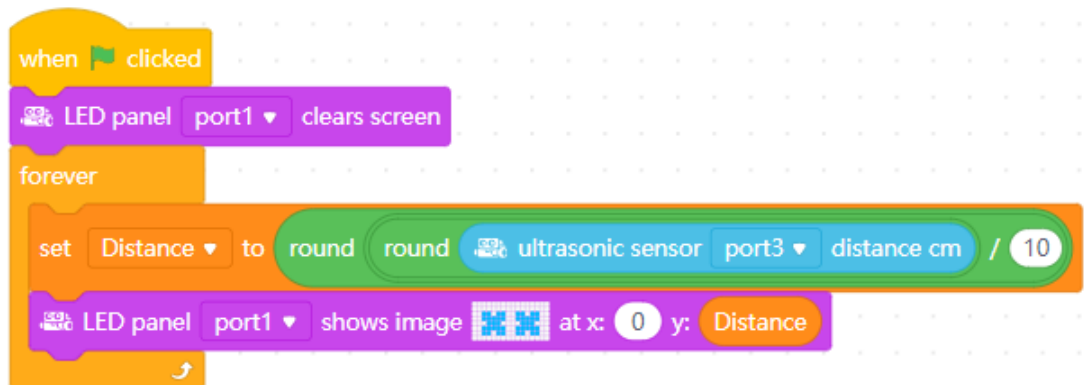
As you can see in the modified script below, you can just put the nested block sequence (that rounds the ultrasonic sensor reading)

directly into the 'shows number' block to get an instant reading on the LED matrix display panel.



This *is* a shorter script, but it does make sense to (and I always do) store feedback data in a Variable; so out of the two scripts above, my preference is for the first one.

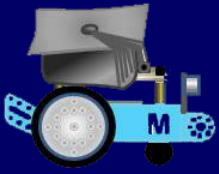
The script on the right uses the 'shows image' stack block used in conjunction with feedback from the ultrasonic sensor to create an animation (of sorts).



This animation script also makes use of the Variable created earlier, called 'Distance' and some complex nesting of blocks.

It uses the 'Reporter' block, 'Round' from the 'Operators' blocks category twice to produce an integer for the y value position which decreases the closer the sensor gets to an object.

The LED Matrix 'Panel' starts by showing nothing and then as the y values decrease it begins to show the image scrolling upward a row at a time. The full image is displayed when 'Distance' = 0 and if 'Distance' (the sensor value) starts to increase once more then the image begins to scroll back down again.



Try experimenting with the script shown on the right.

You need to create a user-defined block (see pages 41, 42 & 64) for more on this). Call it *'Scroll_Text'* and add two input settings when it is defined; a string input called *'Text'* and a number input called *'Delay'*.

```
define Scroll_Text Text Delay
set Counter to 0
repeat length of Text * 6 * 10
  LED panel port1 shows text Text at x: Counter * -1 + 5 y: 0
  change Counter by 1
  wait Delay seconds
```

```
when clicked
set Counter to 0
set Message to Hello mBot Fans
Scroll_Text Message Counter
LED panel port1 shows text clears screen
```

These input settings will give your defined block (*'Scroll_Text'*) two little bubble windows.

Here you can enter the required Variable data, *'Message'* and *'Counter'* (see above).

This is a marginally complex script, but it's definitely worth experimenting with this concept.

Finally, we updated (very early on in our programming journey) Emma's favourite self-programmed games at that time, *"Dice"* and *"Rock / Paper / Scissors"* to take advantage of our newly purchased LED Matrix display panel.

As you will probably be aware by now (from reading Chapters 11 and 12) we did eventually take these games to much higher levels than those outlined below:

To experiment with your LED matrix panel, I suggest that you load the project files you created for these and modify them accordingly. The script shown on the right

```
when up arrow key pressed
set Random to pick random 1 to 6
LED panel port1 shows text Random at x: 5 y: 0
set Text Output 1 to Hey Emma
set Text Output 2 to join You have thrown a Random
```

sets a *'Random'* Variable between 1 & 6 and displays it on the LED Matrix 'Face'. Setting the x value to *'5'*, positions the single random number in the centre of the display. This script (and the one at the top of the next page) also make sensible use of the *'Hat'* block *'when nnn key pressed'*.

The script at the top of the next page is an updated version of our earliest attempts at the *"Rock / Paper / Scissors"* game. Here the script sets another *'Random'* Variable - this time, a random number (between 1 & 3) and based on this number will select from a list a matching graphic to display on the LED panel. The concept of *'Lists'* is featured on page 68. This page describes the use of a *'Useful List'* lookup table containing:

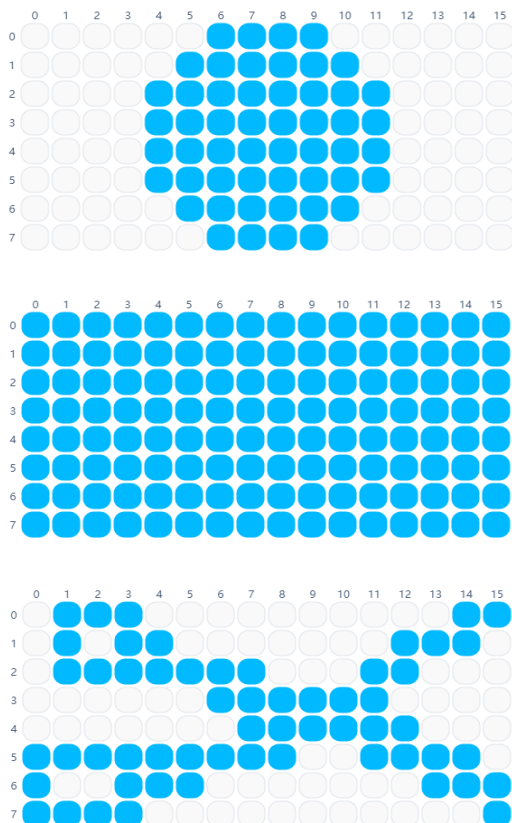
- 1 Rock
- 2 Paper
- 3 Scissors



```

when down arrow key pressed
  set Random to pick random 1 to 3
  set Text Output 1 to Oh Emma!
  set Text Output 2 to item Random of Useful_List
  if Random = 1 then
    LED panel port1 shows image [Rock] at x: 0 y: 0
  if Random = 2 then
    LED panel port1 shows image [Paper] at x: 0 y: 0
  if Random = 3 then
    LED panel port1 shows image [Scissors] at x: 0 y: 0
  
```

The three LED 'Emotion' graphics that you need for both of these "Rock / Paper / Scissors" projects are shown here too:



Another early version of this game (see page 69) was activated by using the ultrasonic distance sensor connected to port3 rather than the 'Green Flag' or a keyboard keypress.

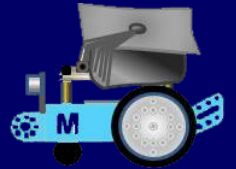
It is not too difficult to add a new Variable called 'Distance' to the script on the left, and then modify it with a 'forever' loop and an 'if ... then' loop.

The nested block sequence needed to poll the sensor is then added and finally a sequence of blocks (needed to wipe everything after a two-second pause) is added to the bottom of the script.

This modified game script is shown below:

```

when clicked
  forever
    set Distance to round ultrasonic sensor port3 distance cm * 10
    if Distance < 200 then
      set Random_Number to pick random 1 to 3
      set Text_1 to Oh Emma
      set Text_2 to item Random_Number of Useful_List
      if Random_Number = 1 then
        LED panel port1 shows image [Rock] at x: 0 y: 0
      if Random_Number = 2 then
        LED panel port1 shows image [Paper] at x: 0 y: 0
      if Random_Number = 3 then
        LED panel port1 shows image [Scissors] at x: 0 y: 0
      wait 2 seconds
      LED panel port1 clears screen
      set Random_Number to 0
      set Text_1 to 
      set Text_2 to 
  
```

The most useful addition to your toolkit is probably the Mini-Spanner (or 'Wrench' as MakeBlock describe it). This complements the excellent double-ended (Posi-drive & 2.5mm Hex) screwdriver that comes with the original mBot kit and most importantly now allows you to tighten up the fastenings on your robot kit without needing to hold the nuts with pliers!

Why, oh why was it not included in the original kit rather than having one distributed in every add-on pack (meaning that I now have four!)? The spanner/wrench has a 5mm A/F slot at one end and a 7mm A/F slot at the other; this end is a little loose, but OK for M4 nuts. There are three more holes for smaller sized nuts situated in the body of the spanner.

In this add-on pack's component parts are several shiny anodised (Makeblock blue) plates, brackets and cuttable linkages. There are two 3-hole x 3-hole Right-Angle Brackets, two 9-hole x 2 Slotted Plates (Code I1) and four 160mm lengths of 20-hole (cuttable) Linkage. There are also four M4*25 brass extender studs (identical to the four that raise mCore above mBots battery pack in the original robot kit). These are so useful for spacing both Me boards and component parts anywhere in a construction.

Also in this pack you get:

- 20 × M4 x 8 Screws
- 10 × M4 Nuts
- 6 × 2mm x 7mm dia. Plastic Spacers
- 2 × M2 x 10 Screws & Nuts (for attaching the Servo to its Bracket Plate)
- 2 × 2.2 x 6.5 self-tapping Screws (for attaching Arms to the centre of the Servo spindle)
- 2 × 2.2 x 8 self-tapping Screws (for attachment of linkages to Servo Arms)

Whilst this is nothing to do with servos, you get one Me series *RGB-LED V1.1* circuit-board module which has 4 little (but full colour) ws2812 RGB LEDs. This is needed for the '*Light-Emitting Cat*'. model. With the modules own integrated chip you can control each little LED individually from your programmes to adjust both brightness and colour by mixing different amounts of red, green, blue light.

Finally, you get the important bit, the **9g Analogue Micro Servo** set (one Servo complete with 3 different *Servo Output Arms*, a rectangular *Servo Bracket Plate* and a circular *Arm Attachment Plate*). Servo output arms are sometimes called '*servo horns*' and have a specific spline size - this pack contains a single-ended horn, a two-point (double-ended) horn and four-point (unequal lengths) horn.

To connect the servo you also get one Me series *RJ25 V 2.2 Adapter* circuit-board which converts mBots standard RJ25 connector to two common signal connectors (for controlling **two** servos). This board contains both a power interface and a signal interface and is used to connect the supplied *Micro Servo* to mCore via a 6P6C RJ25 (20cm) cable. You get two of these cables included in the pack. This adapter can also be used to connect other suitable Me series modules such as the Me Temperature Sensor module.

A servo is a type of motor that can be used in many ways. They consist of a potentiometer and a motor connected to the output spindle, using built in circuitry to accurately control angular movement of the spindle. The output spindle can be positioned to specific angular positions by sending the servo a signal of how far to turn. Small servos have traditionally been used in radio-control applications to position the control surfaces (elevators & rudders) to fly model airplanes or to control the power and steering of model cars.



So, all-in-all, that's a lot of stuff for a tenner or thereabouts and the circuit boards & cables if bought individually would add up to much more than that alone - as I said earlier, it doesn't seem to be well supported with paperwork instructions, but nevertheless this pack is an absolute bargain!

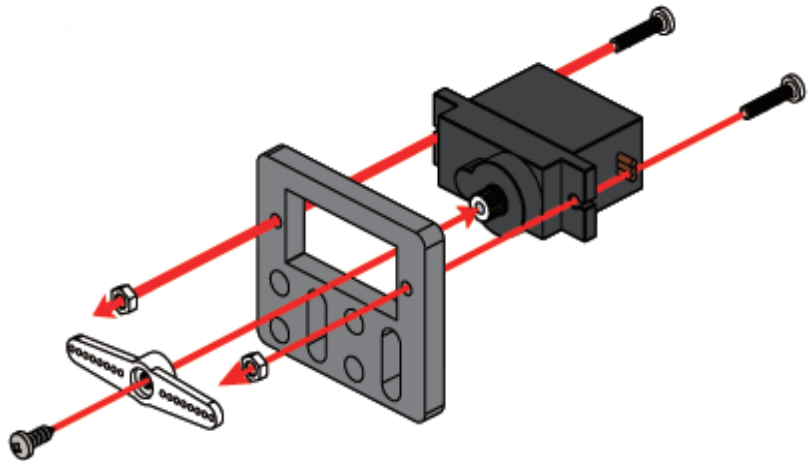
My own design concept for the 'Dancing Cat' model requires a second servo to raise its tail! - (see page nnn). I intended to buy another of these bargain servo packs to double the parts that I have just described on the previous page (the Adapter circuit-board would be surplus to requirement since one board can drive two servos) but at the price, still worth it to gain a second servo and all the other bits!

Sadly, about a month later when I tried to get a second pack they were sold out and unavailable (*I'm not surprised at this price*). Eventually (about two months later) I did get another Servo Pack; this time for £15 inc. UK postage - but I still consider this to be a bargain!

Mounting & Testing the Servo:

Remove the protective paper from the rectangular Servo Plate (which is made from beautifully machined acrylic).

Attach the Servo Unit to the Servo Plate by fitting it into the rectangular recess then bolting it in to place with the two M2 * 10 Screws & Nuts provided. These fasteners are very small and hard to manipulate with large (or small and inexperienced) fingers. For info., the smallest hole in the handle of the spanner is M2 Nut size.



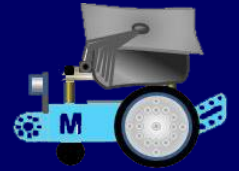
Following the 'Dancing Cat' build instructions (as detailed in Appendix 3)

I started by removing the two screws from the rear of mBot's protective case and replaced them with two M4*25 brass extender studs (see the illustration of the complete assembly at the top of the next page).

Next, I selected a 9 hole x 2 I1 Slotted Plate and bolted a 3-hole x 3-hole Right Angle Bracket to its centre using just two M4 x 8 Screws and Nuts in the frontmost holes only. To this assembly I added a second 9 hole x 2 I1 Slotted Plate mounted on the inside if the angle-bracket with two more M4 x 8 Screws and Nuts. I bolted the RJ25 Adapter circuit board on one side of (and at right-angles to) the first horizontal, I1 Slotted Plate. The adaptor was mounted next to the angle-bracket using just one M4 x 8 Screw and Nut in the hole closest to the angle-bracket (leaving the outer hole free for use later).

I then added two M4*25 brass extender studs to the second (vertical) I1 Slotted Plate using the fourth pair of holes (the ones just above the angle-bracket) securing them in place with two M4 Nuts.

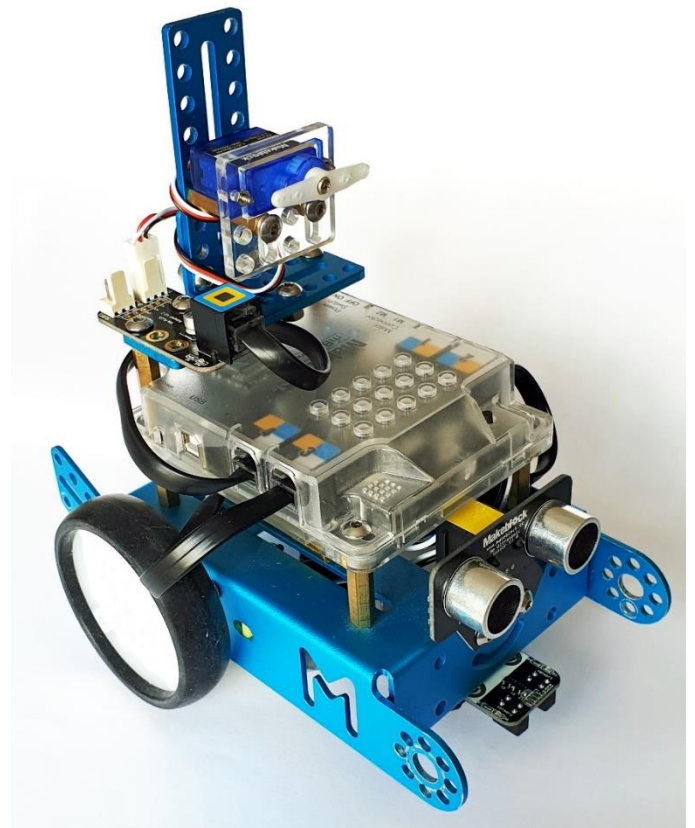
Onto these two extender studs, and using two slightly longer (M4 x 14) Screws, I was able to mounted the Servo Plate; with the Servo body fixed horizontally as shown in the diagram above - the screws passing through the two *slots* in the acrylic plate



The Servo was then connected to its RJ25 Adapter using slot2 (the nearest one of the two connectors to the angle bracket) after winding the length of surplus wire tidily around the upright bracket and plate.

Finally I attached the complete assembly (described on the previous page) to the two M4*25 brass extender studs which I had fitted to the rear of mBot's protective case - the Servo unit pointing forwards over the centre of the case. The left-hand side of the assembly required only an M4 x 8 Screw to attach it to the extender stud, whilst the right-hand stud needed a longer M4 x 14 Screw since it needed to pass through the outer hole in the adapter board as well as the supporting I1 Slotted Plate at the base of the assembly.

The RJ25 Adapter was connected to port4, leaving port2 and port3 connected exactly as they were in mBot's initial configuration and port1 was left empty.



N.B. The servo mounting configuration (shown above right) is exactly as specified for the 'Dancing Cat' model - see Appendix 3 on page nnn)

```

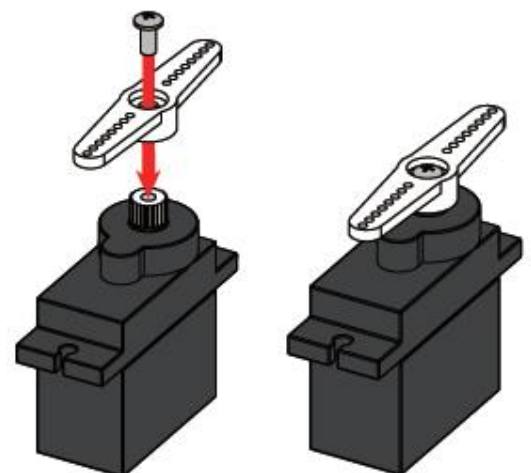
when space key pressed
  servo port4 slot2 positioned at 0
  repeat 3
    wait 0.2 seconds
    play note C4 for 0.25 beats
  
```

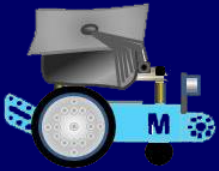
Setting the initial angle for the servo is important, so with the servo now connected to mBot, I created and ran the script shown on the left to set the Servo spindle's angle to zero (0°) degrees.

The servo rotates it's spindle in an anti-clockwise direction up to a maximum rotation angle of 180° degrees. The arm will always return in a clockwise direction to reset itself to zero (0°) degrees after use.

You *must* zero the spindle *before* you fit one of the servo arms. You can then position the servo arm on its splines setting its zero-position angle according to the nature of the task that you intend to carry out.

So at this point, *and only now*, should you attach one of the white Servo Arms supplied (as shown in the diagram on the right). Use a 2.2 * 6.5 self-tapping Screw to attach the arm to the centre hole in the Servo spindle (once again, these are very hard to manipulate with large (or very small and inexperienced) fingers.





mBot and Me a Beginner's Guide

- Remember, the Micro Servo unit can rotate its Arm through approx. 180° and no further.
- Do beware too that the 9g Analogue Micro Servo supplied in this add-on pack is not as strong as a standard radio-control servo (but its fine for mBot models if treated carefully).
- The Arm only grips the Servo's spindle by small splines in the plastic, so it can be damaged (as can the Servo motor itself) if too much force or leverage is applied.

I fitted the a two-point, double-ended, horn (shown at the bottom of the previous page). I suggest that this is the arm to use for most model options and that you centre it either horizontally across the middle of the Servo (parallel to the top of the Arm-Attachment Plate) or vertically (at right angles to the Plate).

After initial calibration, you should test the Servo with a simple programme similar to the one shown on the right; to check that the servo arm moves as expected:

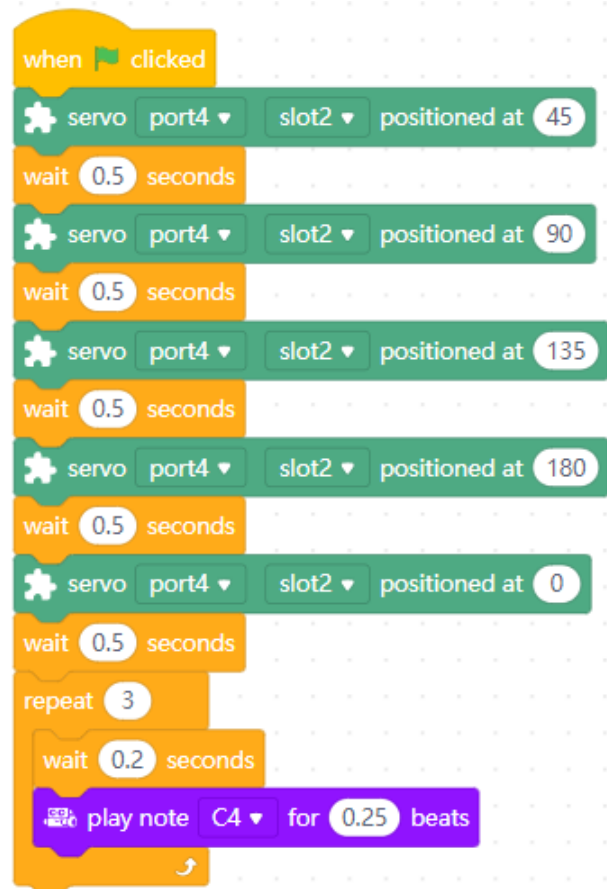
I was by now keen to start building and programming the much anticipated 'Dancing Cat' model (Emma LOVES cats!). This model also uses the two-point (double-ended) horn and the instructions suggest that the horizontal position of the Servo Arm should be set not to 0° at but at 90°.

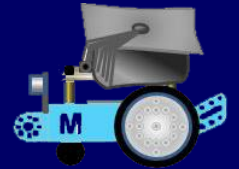
As mentioned earlier, with little information in the pack (other than the 'Dancing Cat' build instructions) I tried to find out more but finding detailed information on-line about Makeblock's add-on packs was frustrating. Typically, the Makeblock forum pages often show mBot owners individual frustrations (which often elicited no response at all!). One such comment posted was:

"Our daughter received her first expansion pack today and there are only directions for one option, the dancing cat. She wants to do the head-shaking cat, but there are no directions. Can you please help?"

Eventually after much on-line searching and false trails I found the model making instructions for 'Dancing Cat' and the missing instructions for the other Servo Pack models (the 'Head-Shaking Cat' and the 'Light-Emitting Cat'). At the time that I was looking (mid-2018) these were well hidden in the Makeblock website; but they are now more easily identified, and you can find these files fairly easily yourself using the following link:

<http://learn.makeblock.com/en/mbot-add-on-packs/>





This link leads you to a web-page about mBot add-on packs which also has building / programming instructions for other add-on packs too (see the illustration on the right):

The programming files are all stored as **.rar** files (which I guess might have been off-putting for some users in the past?). *A .rar file is a compressed file that holds other files inside it compressed down to a much smaller size.* These need special software to open and extract the contents, nevertheless I went ahead and downloaded the Servo Pack 'Program.rar' file.

The building instruction files are in **.pdf** format, so I also downloaded the Servo Pack 'Building Instruction.pdf' file which shows details about servo assembly & use and then (in the following order) how to modify mBot to make three distinct models: (1) 'Dancing-Cat', (2) 'Head-Shaking Cat' and (3) 'Light-Emitting Cat'.

I decided to use the highly-regarded and much-recommended open-source programme '7-Zip' which is totally free and a good utility for unpacking many varieties of compressed format files. So (and with some trepidation) I downloaded it and used it to extract the .rar files. There were no problems with this download and the software was easy to understand so decompression of the .rar files worked well.

The downloaded package of programming files was entitled:

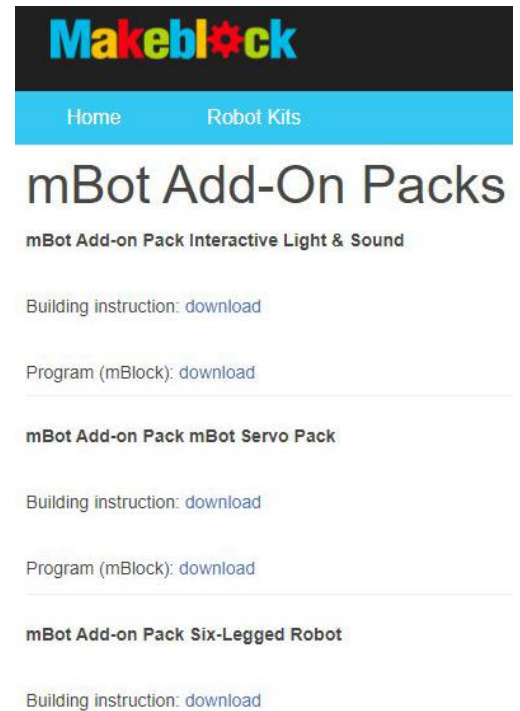
mBot 动感扩展包_程序示例.rar which (using Google Translate) equated to **MBot dònggǎn kuòzhǎn bāo_chéngxù shìlì** and translated as: “ **mBot Dynamic Expansion Package _ program example.rar** ”.

When the file was processed by '7-Zip' it extracted a folder with the same name (mBot 动感扩展包_程序示例) and this folder contained three **mBlock 3** (.sb2) project files - but were they in the same order as the building instructions? On examining these mBlock 3 files I was delighted to see that the script blocks were the English language versions, but the comments call-outs describing them *were* in Chinese.

I was probably hampered a bit with the Chinese to English translations of both the filenames and the comments contained within the files and in retrospect I now know that the contents of the downloaded **Program.rar** file, are not in the same order as the downloaded Building Instruction .pdf file.

I was now able to load these three .sb2 files into mBlock 3 and edit the comments. The first comment call-out from the first 'The Kitten Looking Around' file that I translated had the following flowery style: “*This program according to the angle of the servo installation is different, need to adjust the servo angle value*”. Not over-clear, but enough for me to work on to create rather more understandable English translations. The translated comments for all three models were all very much in a similar vein, but some more understandable than others.

The three files extracted from the download are itemised on the next page, whilst the mBot modifications required to make the three 'Cat' model projects (the building instructions) and the necessary mBlock 5 programmes to control them are described in the next three appendices.





File 1: 东张西望的小猫.sb2 - which equates to: **Dōngzhāngxīwàng de xiǎo māo.sb2** and translates as: **'The Kitten Looking Around.sb2'**

This file contained a large complex script headed by an *'mBot Program hat'* block (signifying uploading it to Arduino for *'Off-line'* use). It has a 'disco' stage backdrop & a single sprite (named 'mBot'). The false impression given by the *'Stage'* background graphics - 'a disco, with lights' confused me!

It took me some time to work out that this file was actually the **'Head-Shaking Cat'** programme (although I thought on first viewing that it might, thanks to the 'disco' lights look, be related to 'light-emitting'). This was the **second model** in Makeblock's sequence of building instructions

File 2: 小猫探照灯.sb2 - which equates to: **Xiǎo māo tànzhàodēng.sb2** and translates as: **'Kitten Searchlight.sb2'**

'Searchlight' was the clue here! This seemed to be the **'Light-Emitting Cat'** programme and the file had *lots* of scripts which were mostly activated by keyboard single keypresses; but there were two slightly longer scripts and some comments referred to a 'tail' and 'lights': "*Press the space bar, the kitten rotates to the right and swings the tail*" and "*Release the space bar, the kitten stops rotating, the servo angle is reset, the light goes out*". This was the **third model** in the sequence of building instructions.

File 3: 跳舞的小猫.sb2 - which equates to: **Tiàowǔ de xiǎo māo.sb2** and translates as: **'Dancing Kitten.sb2'**

This seemed to be the **'Dancing Cat'** programme, the names matched, although the file contained just one short programme (also starting with an *'mBot Program hat'* block). There was a self-defined block script called 'servo' which used a repeat loop to swing the arms 5 times and a comment: "*If someone is close, and the distance is less than 20cm, then it swings the arm and turns left and right in welcome*". This was the **first model** in Makeblock's sequence of building instructions.

Now, in 2019, and with totally new mBlock 5 software I have had to totally rewrite the files and whilst these owe much to the original concepts, they do look totally different and I have added (as you will see) several modifications of my own.

Why Cats? ...

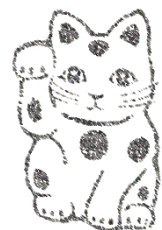


no copyright
infringement
intended

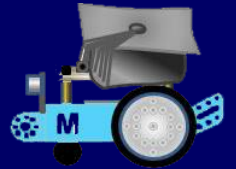
... Is this perhaps a reference to the so-called "*Chinese Lucky Cat*", very popular with Chinese merchants in many countries?

The "*Chinese Lucky Cat*" is a cat figurine, often with an automated slow-moving single paw, which waves and beckons customers; these are usually displayed at the entrance of shops, restaurants and other businesses.

This figurine is however *Japanese*, and it's called the "*maneki-neko*" the '*beckoning cat*', which is a common Japanese lucky charm or talisman believed to bring good luck to the owner.



no copyright
infringement
intended



Appendix 3 - mBot Servo Project - 'Dancing Cat'

Building / modifying the 'Dancing Cat' model

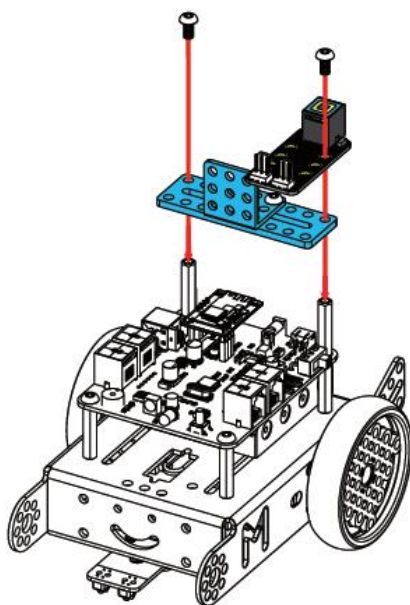
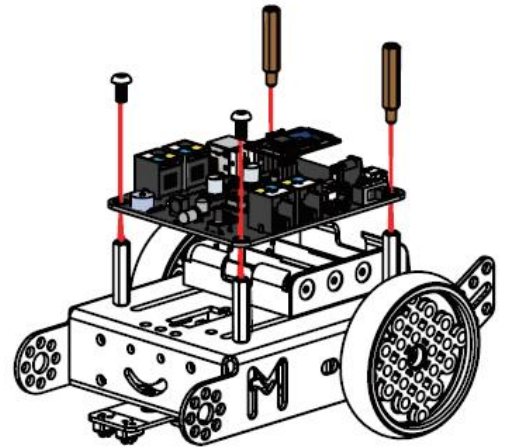
I have already outlined my version of modifying mBot to build the 'Dancing Cat' mBot model in Appendix 2, but the enclosed instructions start by suggesting that you disconnect the cable and remove the Ultrasonic Sensor from the front of mBot to create the "Johnny 5" look (he's the robot in the film "Short Circuit").

You can do this if you want to, but I quite like leaving the sensor where it is - this makes less work to do if you want to switch models frequently and the concept of moving the sensor to the top of a 'neck' (which you are about to build on top of mBot) is only to suggest that the Ultrasonic Sensor receptor tubes are the 'cats eyes' above the 'waving paws' - and I had a much better idea for the 'paws'!

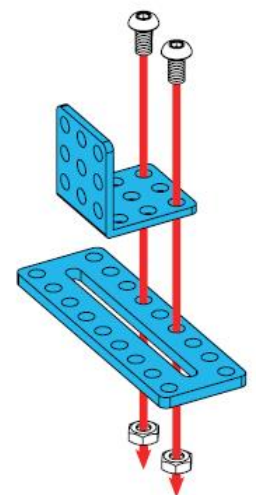
The instructions show undoing the four screws holding the cover and removing the protective cover from the top of mBot. I can't see the point of doing this either. Once again, this is less work to do and the whole construction sequence shown below works fine with the cover still in place protecting the mCore board.

If you do follow my guidelines, then just add the two new brass extender studs to the rear as described in Step 1 below ignoring the instruction to remove the front screws.

Step 1. Replace the two rear screws holding mCore and its cover down on to the rear two brass pillars and add two additional brass pillars (shown in the diagram above right) to secure mCore to the two lower brass extender studs.



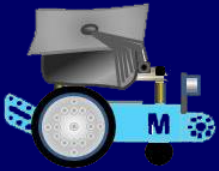
Step 2. Assemble one of the 3 x 3-hole Right Angle Brackets with one of the 9-hole x 2 (Code II) Slotted Plates (as shown in the diagram on the right) using two nuts and bolts - note which holes are used for this.



Step 3. Use two screws to attach your assembly to the top of the two new rear brass extender studs (as shown in the diagram on the right).

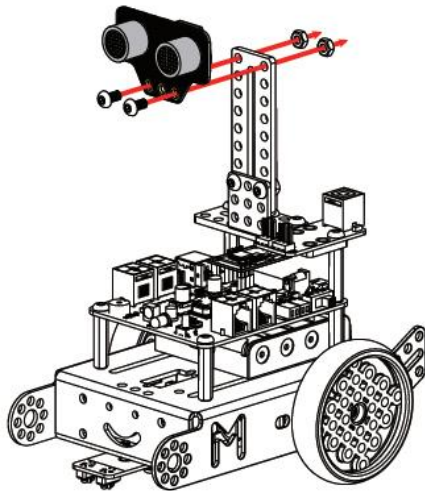
Then attach the Me RJ25 Adapter board to the right of the angle bracket with two more screws. (N.B. One of these should be the screw that goes down into the lower brass extender stud - this is shown in the diagram on the left).

The second screw through the adapter board will need a nut to secure it on the underside of the slotted plate.

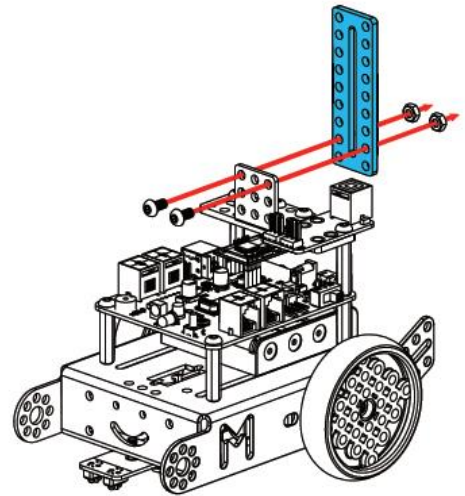


mBot and Me a Beginner's Guide

Step 4. Attach the second Slotted Plate vertically to the right-angle bracket component that you have just added to your new assembly. Use two more M4 Screws and Nuts to do this (note from the diagram on the right which holes to use).

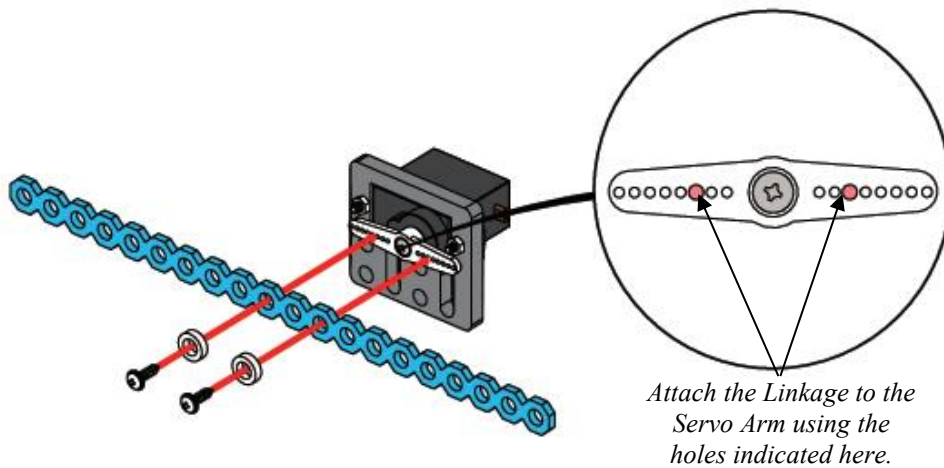


Step 5. Unless you have followed my suggestion of not moving it, re-attach the Ultrasonic Sensor you removed earlier to the top of your new assembly using two more screws and nuts (as shown in the diagram on the left) - the "Johnny 5" look!



Step 6. If you haven't done it already (and you probably have by following my notes on page 194), assemble the Servo and its Servo Plate. Test the Servo, rotating it to a setting of 90° and then (and only then) position the Servo Arm horizontally across the Servo unit.

Step 7. Attach one of your 20-hole (cuttable) Linkages to the Servo Arm using two 2.2 x 8 self-tapping screws passing through two of the 2mm plastic ring spacers (see the diagram below):



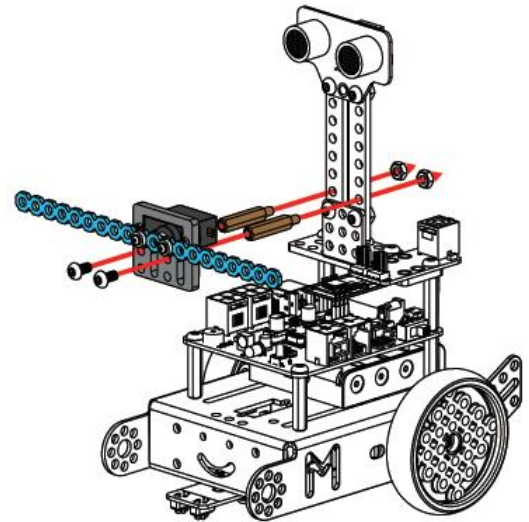
Attach the Linkage to the Servo Arm using the holes indicated here.

Beware, it's very easy to drop these little screws, they are so tiny for big fingers and you only have 2 (and no spares!). I didn't find the supplied screwdriver the best fit into the cross-head of these screws, so I used another (a jewellers screwdriver) from my collection. I do strongly recommend screwing each screw into the *third-hole-out* from the centre first, so that it taps a thread into the hole in the servo arm and thus becomes slightly easier to fit a second time when you attach the Linkage later.

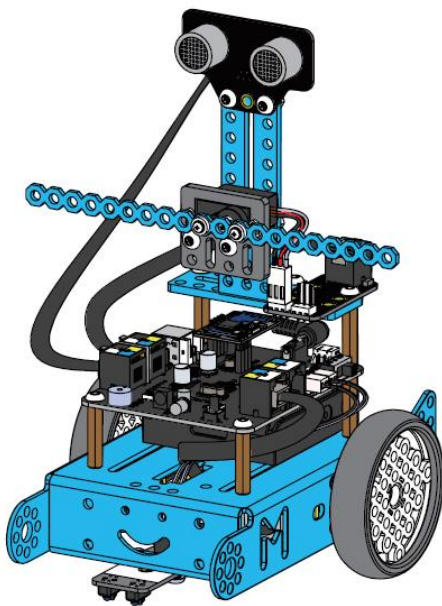
Cuttable Linkages have 20 holes, therefore there is no centre hole equidistant from each end. I suppose that you could cut off a single hole unit from one end leaving 9 holes evenly spaced either side of a central hole (and leaving a single hexagonal washer). But cutting a length of Linkage isn't really necessary for this model, since it still works even though the arm is slightly unbalanced. *I did however decide to cut one length of Linkage into two 9-hole lengths & 2 hexagonal washers.* (See page 200).



Step 8. Attach your Servo / Linkage assembly to the remaining two brass extender studs with two screws as shown. Attach the two brass extender studs and Servo / Linkage assembly to the vertical slotted plate on top of mBot using two M4 Nuts as indicated in the diagram on the right. - note which hole positions are used for this.



Step 9. It's time to connect-up the Me modular components to mCore. The Ultrasonic Sensor can be reconnected using an 6P6C cable to the RJ25 Me socket where it was connected in the original mBot configuration (port3).



Connect the Me RJ25 Adapter circuit-board to port4 using another 6P6C RJ25 Me cable.

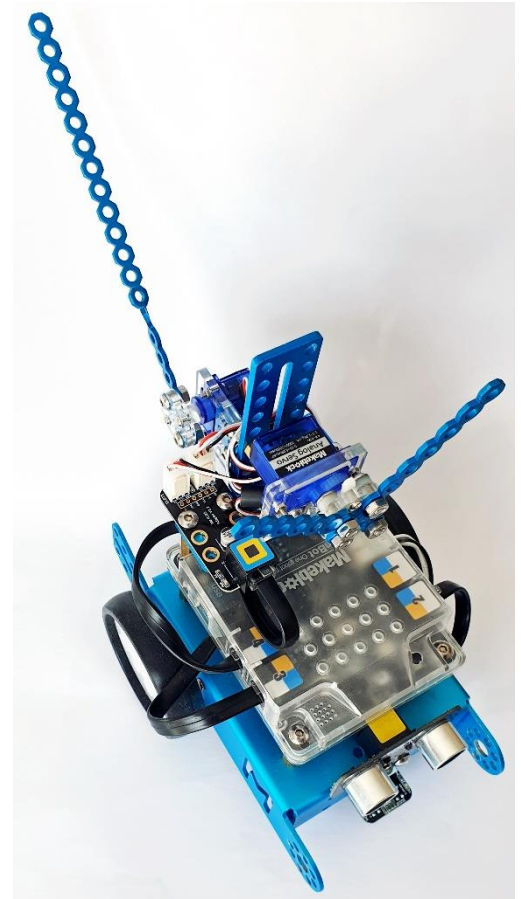
You cannot use any of the other mCore ports for this project since this module requires a black coded port and the other (port3) is needed to power the ultrasonic sensor.

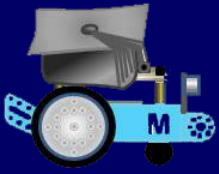
You also need to connect the Servo to slot2 of the RJ25 Adapter board - slot2 is the left-hand one of the two signal connectors when looking at the end with these two connectors. My own 'test rig' variant on the theme of the 'Dancing Cat' model is shown in Appendix 2 (on page 195).

Not liking the full length of Linkage attached to the

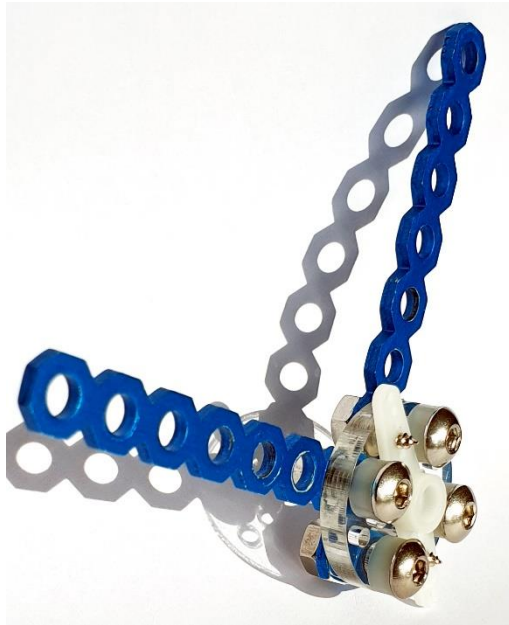
servo arm - the 'see-saw' arm of the original model concept (shown above) I decided to cut one 20-hole length of Linkage into two short 9-hole lengths and I then cut the remaining left over (two-holed) piece into two octagonal washers, knowing that they would be useful as spacers later. The 'cuttable' Linkage was indeed very easy to cut with a junior hack-saw was easy to clean up too with a small fine file too.

I removed the full length of Linkage which had been mounted directly onto the Servo Arm and replaced it using the same screws and the same mounting holes with the small (20mm dia.) circular acrylic plate that came with the Servo pack. The head of the central screw holding the Servo Arm to the splined shaft of the Servo seemed to be raised slightly, so I used a 4.5mm drill to enlarge the central hole in the acrylic plate to clear this. The enlarged hole also allows the screw to pass through the plate when you need to unscrew it again.





mBot and Me *a Beginner's Guide*

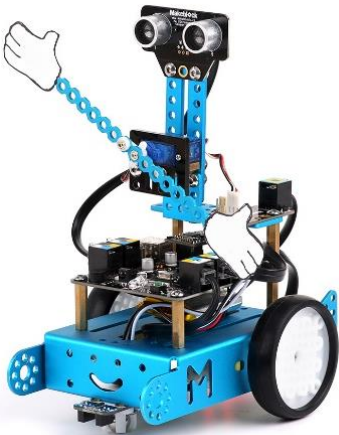


I then fitted one of my newly cut 9-hole short Linkage lengths to the circular acrylic plate using two M4 x 15mm Screws, two 2mm plastic spacers and M4 Nuts, using the first and third holes at one end.

Positioning the second cut Linkage piece on top of the first one (with a 90° angle between the two) I used two M4 x 15mm Screws, two 2mm plastic spacers and M4Nuts together with my two little hexagonal offcuts as spacer pieces between the second *top* short Linkage arm and the plate (but these could have been two more of the 2mm plastic spacers instead). See the illustration of the modified 'arms' on the left.

This configuration now looked so much more like waving arms than the single see-saw Linkage that they were replacing!

I had seen somewhere on a website the 'Dancing Cat' model with two small 'hands' cut out from cardboard and attached to either end of the full length of 20 hole Linkage attached to the Servo Arm (see a picture of this below).



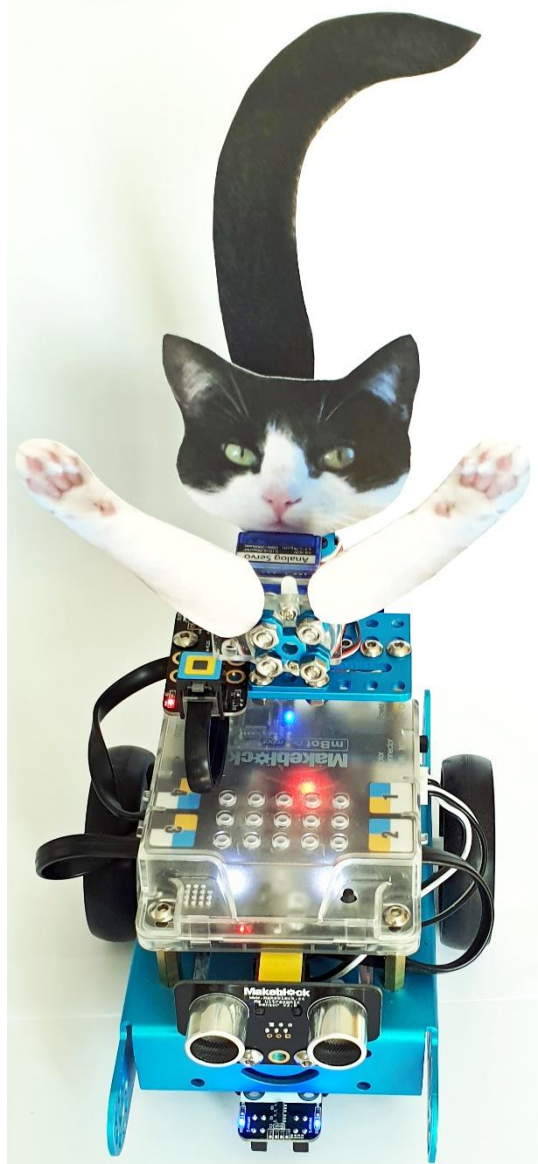
no copyright infringement intended

I thought that I could make a much better attempt at this and took a suitable picture of Emma's cat 'Ko-Ko'. I edited the picture extracting a 'head shape', a 'tail shape' and a 'paw / leg' shape (which I duplicated and 'flipped' to make a second paw).

I made the head about 70mm wide, the two paws about 70mm long and the tail about 120mm long. I then printed

them on to cardboard. Emma really did like this a lot and she had the task of cutting out the pictures and gluing them in place. Using 'Glue-Dots' as fixings the illusion was completed by the cardboard head being attached to the top of the vertical slotted plate *above* the servo; the paws were attached to the two cut lengths of Linkage on the front servo and the tail to a piece of twisted Linkage on the rear servo.

This really was a much more realistic 'cat-bot' when compared to the original ("Johnny 5 with gloves on") configuration shown in the picture above.





Programming the 'Dancing Cat' model

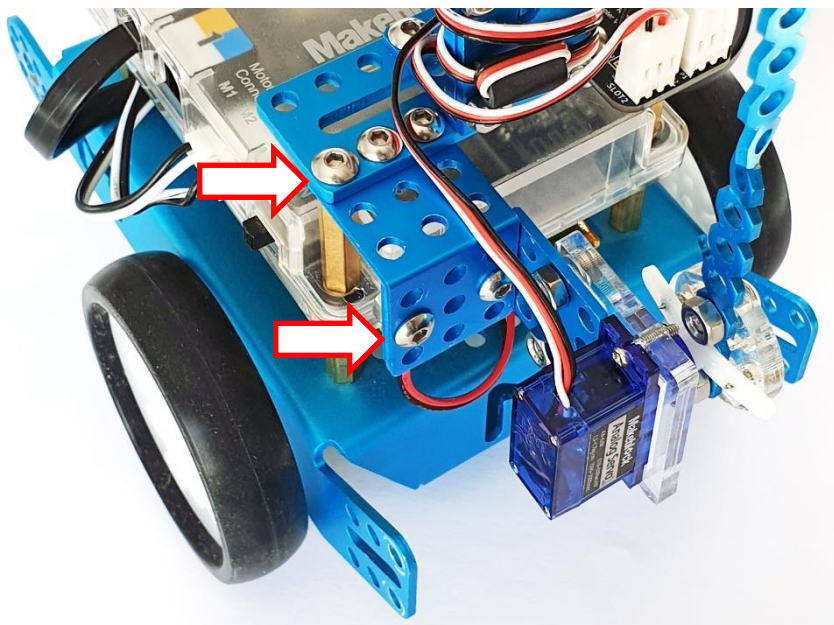
Modifications to mBot for the 'Dancing Cat' model very nearly completed; it was time to programme it.

Project Concept: *The 'Cat' sits waiting but raises its 'tail', 'meows', waves its 'paws' and dances excitedly from side-to-side in welcome if it detects someone close to it.*

Shown on the right is a Flow Chart algorithm that I created to analyse the logical processes required to achieve the objectives of the above concept:

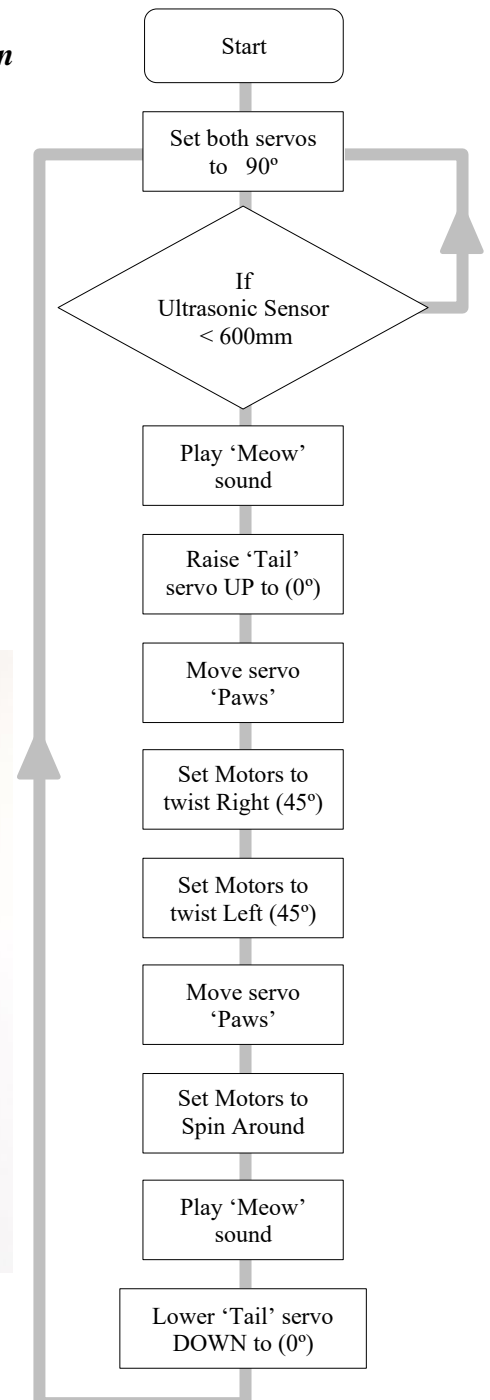
A cat raising and lowering its tail is a very common gesture of greeting. Since I had now bought two Servo packs I was able to fully utilise the second of the signal connectors on the adaptor board (now attached to mBot) by adding my second servo to emulate this tail movement.

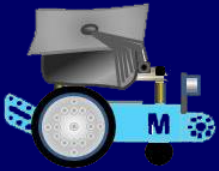
It was fairly straightforward to attach two more right-angle brackets to the rear of the servo assembly described earlier. This modification is shown in the illustration below:



The second Servo was attached to slot1 on the adaptor board and the servo arm mounted vertically on its splined shaft.

To attach the 'tail' I took one length of 20-hole Linkage and between the 7th & 8th holes from one end twisted it through 90° so that a cardboard tail could be added to it with another couple of 'Glue-Dots'. The twisted linkage was then attached to the arm of the servo using the second circular acrylic plate. Shown on the next page is the complete script sequence written to fulfil both the project concept and the processes outlined in the Algorithm shown above.





mBot and Me a Beginner's Guide

My modified and adapted 'Dancing Cat' programme (on the 'Devices' tab):

```
when clicked
  Reset_Ports
  wait 0.5 seconds
  forever
    Set_Distance
    if Distance < 600 then
      broadcast PPlay_Meow
      Show_Interest
      broadcast PPlay_Meow
      Reset_Ports
    else
      stop other scripts in sprite

define Show_Interest
  servo port4 slot2 positioned at 135
  Raise_Tail
  Wave_Paws
  Twist_Right
  Twist_Left
  Spin_Around
  Lower_Tail
  servo port4 slot2 positioned at 45

define Spin_Around
  turn right at power 100 % for 1.25 secs

define Twist_Right
  turn right at power 100 % for 0.1 secs
  turn left at power 100 % for 0.1 secs

define Twist_Left
  turn left at power 100 % for 0.1 secs
  turn right at power 100 % for 0.1 secs

define Wave_Paws
  servo port4 slot2 positioned at 130
  wait 0.2 seconds
  servo port4 slot2 positioned at 50
  wait 0.2 seconds
  servo port4 slot2 positioned at 90

define Raise_Tail
  servo port4 slot1 positioned at 0

define Lower_Tail
  servo port4 slot1 positioned at 90

define Reset_Ports
  servo port4 slot1 positioned at 90
  servo port4 slot2 positioned at 90

set Distance to round ultrasonic sensor port3 distance(cm) * 10

... & on the 'Sprites' Tab
when I receive PPlay_Meow
  start sound Meow
```

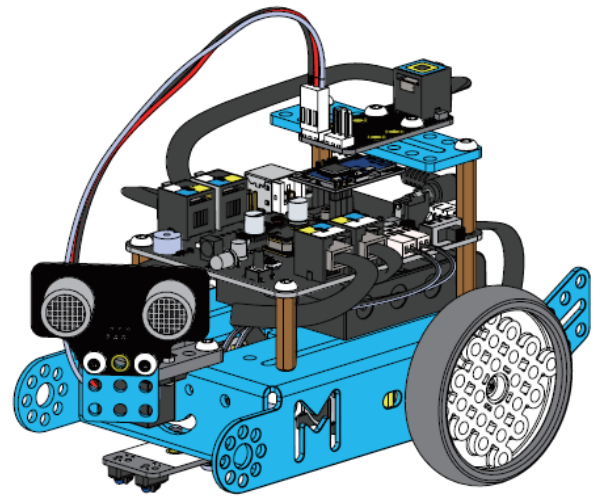



Appendix 4 - mBot Servo Project - 'Head-Shaking Cat'

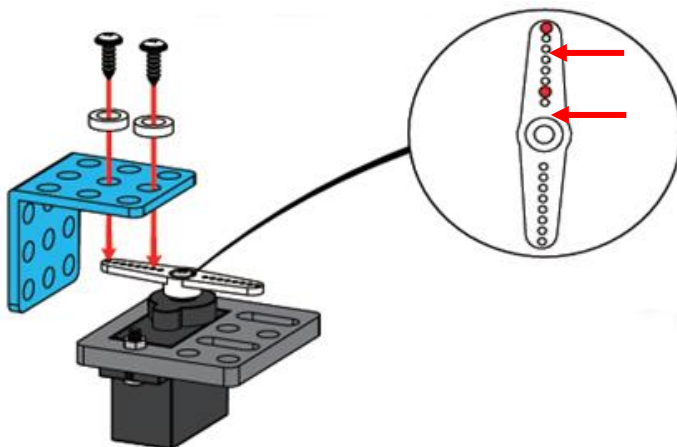
Building / modifying the 'Head-Shaking Cat' model

The 'Head-Shaking Cat' mBot model is much simpler to create than the 'Dancing Cat', but some aspects of it are very fiddly and once again, in some places, awkward for big (and rather old) fingers.

If you are moving on from my 'Dancing Cat' model described earlier in Appendix 3 (and I am assuming that you are) then you need to disconnect and then unbolt the front-facing Micro Servo Plate from the two brass studs holding it to the upright slotted-plate at the back of mBot.



Unbolt the 3 x 3 hole angle-plate and the upright slotted plate that held the servo too. The horizontal slotted plate at the back of mBot can be left in place, as can the RJ25 Adapter Module, leaving it attached and ready for reconnecting the servo later.



This Micro Servo unit needs to remain in its mounting plate (exactly as before with the 90° position of the splined spindle set at the top).

The servo plate will be mounted horizontally, not vertically as it was for the 'Dancing Cat' model.

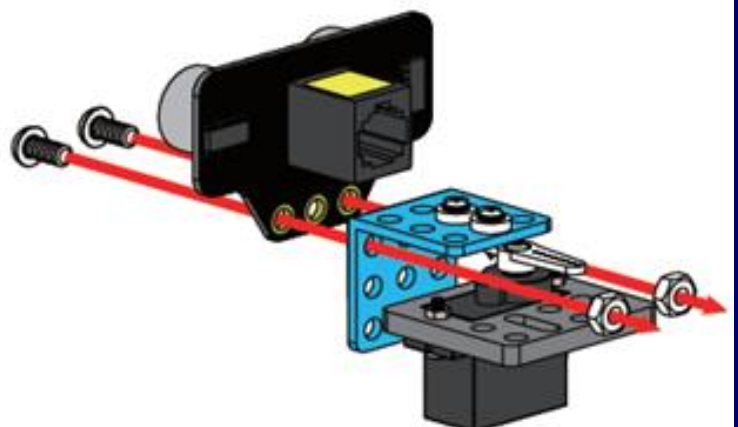
A 3 x 3 hole angle-plate needs to be attached to the servo arm next. I would suggest that *it's much easier* to fit the angle bracket to the top of the servo arm using two M2 × 8mm self-tapping screws and 2 spacers (as shown in the diagram

on the left) *before* the arm is fitted back onto the servo. I struggled with this quite a lot (*it is MUCH harder to fit these little screws than it looks here!* Screw them both into the indicated holes in the servo arm first to tap the thread into the plastic before fitting the washers and attaching the arm to the bracket).

Note the servo arm position shown in both diagrams. Remove the Ultrasonic Sensor unit from the front of mBot and disconnect its RJ25 cable.

Attach the sensor unit to the angle bracket / servo assembly using two M4 × 8mm Screws and M4 Nuts (as shown in the diagram on the right).

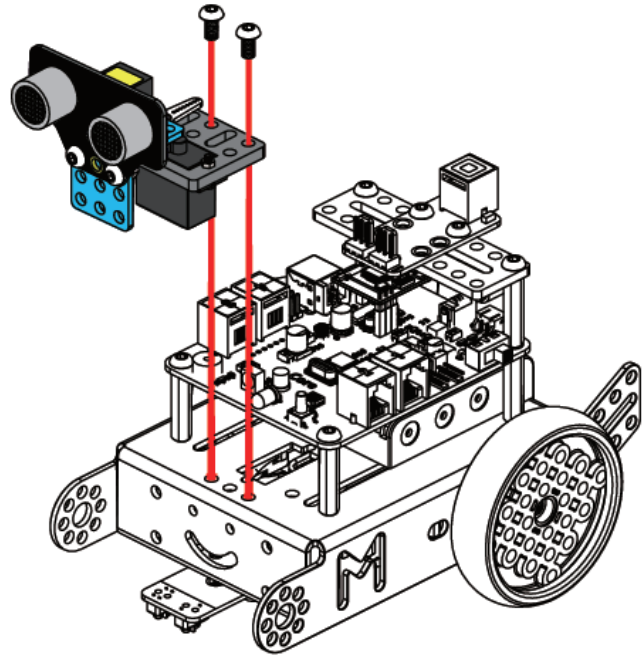
This is also slightly harder to do than it looks.





I used a small pair of snipe nosed pliers to hold the nuts whilst the screws were tightened with the hex-drive screwdriver.

Attach the servo / sensor assembly that you have just created to the front of mBot using two M4×8mm screws (and nuts if necessary). This is also much harder to do than it looks, particularly if the chassis on your mBot doesn't have threaded holes on the top into which you can just insert the screws. Holding nuts underneath whilst rotating the screws is *very* tricky, so I using longer screws fitted from underneath and held in place with the screwdriver whilst the nuts were added on top of the servo plate (using the slots in the plate and not the holes indicated on the right) made this easier too! Note the assembly diagram shown on the right.



That's just about all that is required to modify mBot to make this model. Beware though, the attachment of the Ultrasonic Sensor to the servo arm (which I fitted LAST) is a rather flimsy affair and on my model the Sensor drooped downwards slightly (due I guess to its weight and consequent leverage on the thin plastic servo arm). Having said that though, it seemed to work quite well.

All you need to do now is to remake the RJ25 connections required for this model. The ports used in the downloaded demonstration programme use port3 for the Ultrasonic Sensor and port4 for connecting the Servo Adapter module.

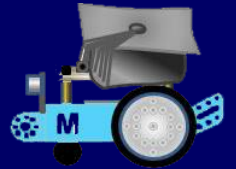
Since the '*Dancing Cat*' model used port4 for the Servo Adapter then there is nothing to change here.

The initial setup of mBot had the Me Ultrasonic Sensor connected to port 3 (as used in the '*Dancing Cat*' model) so I suggest keeping that connection and adapting the programme accordingly, leaving the line follower module connected to port2 whilst port1 remains empty.

Programming the '*Head-Shaking Cat*' (mk. 1) model

Construction of the '*Head-Shaking Cat*' model completed; it was time to think about transcribing the '*The Kitten Looking Around*' Makeblock programme file that I had downloaded for this project. This was the rather complex looking (mBlock 3.sb2) file described in Appendix 2 on page 198 and shown in its entirety on the left-hand side of the next page. The mBot model worked very well using mBlock 3.

Transcribing a new version this file in mBlock 5 was not too difficult and considerably easier than I thought it might be. It worked the first time that I tried it, operating the mBot model in exactly the same way as if using the original script. The transcribed **.mblock** file is shown on the right-hand side of following page and as you will see, they do look very similar.



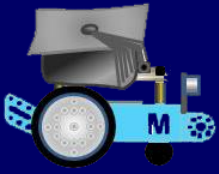
```

mBot Program
set speed to 100
play tone on note C4 for 0.25 beats
set led on board all red 60 green 0 blue 60
set servo port4 slot2 angle 90
wait until on board button pressed
play tone on note C4 for 0.25 beats
set led on board all red 60 green 60 blue 0
forever
run forward at speed 60
wait 0.5 secs
set servo port4 slot2 angle 180
set led on board all red 0 green 0 blue 60
wait 0.5 secs
set 0.1 to ultrasonic sensor port3 distance
set servo port4 slot2 angle 100
set led on board all red 60 green 0 blue 0
wait 0.5 secs
set 0.2 to ultrasonic sensor port3 distance
set servo port4 slot2 angle 90
set led on board all red 0 green 60 blue 0
wait 0.5 secs
set 0.3 to ultrasonic sensor port3 distance
set servo port4 slot2 angle 90
set led on board all red 60 green 0 blue 0
wait 0.5 secs
set led on board all red 0 green 0 blue 0
if 0.2 < 25 then
set led on board all red 60 green 0 blue 0
run backward at speed speed
wait 0.5 secs
if 0.3 - 0.1 > 10 then
turn right at speed speed - 20
else
turn left at speed speed - 20
else
if 0.1 < 15 and 0.3 - 0.1 > 10 then
set led on board led left red 60 green 60 blue 0
turn right at speed speed - 20
else
if 0.3 < 15 and 0.3 - 0.3 > 10 then
set led on board led right red 60 green 60 blue 0
turn left at speed speed - 20
else
set led on board all red 60 green 60 blue 0
run forward at speed speed
wait 0.5 secs
    
```

```

when clicked
set Set Speed to 75
all play note C4 for 0.25 beats
all turn on all light with color red 60 green 0 blue 60
wait until all when on-board button pressed
all play note C4 for 0.25 beats
all turn on all light with color red 60 green 60 blue 0
forever
stop moving
wait 0.5 seconds
servo port4 slot2 positioned at 180
all turn on all light with color red 0 green 0 blue 60
wait 0.5 seconds
set Range_Left to round all ultrasonic sensor port3 distancecm
servo port4 slot2 positioned at 90
all turn on all light with color red 60 green 0 blue 0
wait 0.5 seconds
set Range_Centre to round all ultrasonic sensor port3 distancecm
servo port4 slot2 positioned at 90
all turn on all light with color red 0 green 60 blue 0
wait 0.5 seconds
set Range_Right to round all ultrasonic sensor port3 distancecm
servo port4 slot2 positioned at 90
all turn on all light with color red 60 green 0 blue 0
wait 0.5 seconds
if Range_Centre < 25 then
all turn on all light with color red 60 green 0 blue 0
all move backward at power Set Speed %
wait 1.5 seconds
if Range_Right - Range_Left > 0 then
all turn right at power Set Speed %
else
all turn left at power Set Speed %
else
if Range_Left < 15 and Range_Right - Range_Left > 0 then
all turn on left light with color red 60 green 60 blue 0
all turn right at power Set Speed - 25 %
else
if Range_Right < 15 and Range_Left - Range_Right > 0 then
all turn on right light with color red 60 green 60 blue 0
all turn left at power Set Speed - 25 %
else
all turn on all light with color red 60 green 60 blue 0
all move forward at power Set Speed %
wait 0.5 seconds
    
```

In the process of transcribing the mBlock 3 file into mBlock 5 format, these long sequences of nested decisions became easier to understand which allowed me to break it down into much smaller scripts.



mBot and Me

a Beginner's Guide

The concept of the original Makeblock (mBlock 3) programme appeared to be as follows:

The 'cat's head' starts off looking forwards (the sensor in the middle position). When the programme is activated, the 'cat' moves forwards rather hesitantly for approx. one second, it then pauses to 'look' first right, then centre and then left by moving the sensor from side to side using the servo.

If no contact is detected by the sensor (within a given range) then the 'cat' continues to move forwards as before. When the moving 'head' detects a contact it stores the Range in a variable) then it makes a decision of what to do next based on pre-defined 'a contact is close' or 'a contact is very close' parameters.

If the head turns to the left and detects something within range then the 'cat' backs away and turns right. If the head turns to the right and detects something then the 'cat' backs away and turns the opposite way (to the left).

To emulate this in mBlock 5 I broke up the original script that I had transcribed into nine clearly descriptive self-defined 'My Blocks' scripts, named as follows: 'Head_Feedback', 'Make_Decision', 'Move_Backwards', 'Move_Forwards', 'Move_Timer', 'Pause_Timer', 'Poll_Range', 'Turn_Left' and 'Turn_Right'.

I also created three Variables to store the readings from the Ultrasonic Sensor every time that it was polled. These were sensibly named as: 'Range_Centre', 'Range_Left' and 'Range_Right'.

I removed all of the blocks from the original script that played tones through mBot's buzzer (apart from one which I left to signify that the programme was activated and that mBot was ready). I also cut out most of the blocks referring to mBot's on-board lights and the one block setting-up mCore's on-board button.

Thanks to these deletions and my use of self-defined blocks, the primary activation script (shown here on the left) now became a very short and much clearer and understandable sequence.

You will note that this '**mBot start**' script is activated by pressing the **Zero (0) key** on the keyboard. On most keyboards, the numeric keypad usually has a large **Zero (0)** key just to the right of the cursor keys and on the same line as the **Spacebar**; so setting this keypress to start mBot is a sensible choice.

Similarly, using a simple '**mBot stop**' script to halt the '**Head-Shaking Cat**' model is easy too if you make the actioning keypress the **Spacebar** at the bottom of the keyboard. The spacebar is so easy to hit in a crisis should you need to stop mBot.

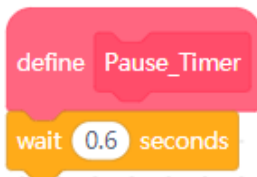
This script is shown on the next page along with four simple scripts which will enable you to manoeuvre mBot manually using the four cursor keys on the keyboard.



On the right, is the **'mBot stop'** script discussed on the previous page.

The remaining scripts here enable you to control mBot manually via the cursor keys on the keyboard. The 'up arrow' cursor key is **'mBot forward'** whilst the 'down arrow' cursor key is **'mBot reverse'**. The cursor key 'left arrow' is **'mBot left'** and the cursor key 'right arrow' is **'mBot right'**.

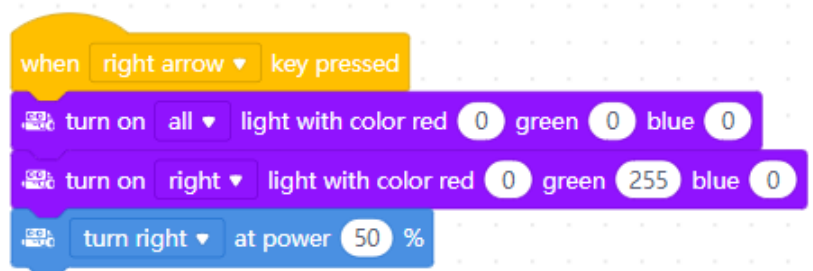
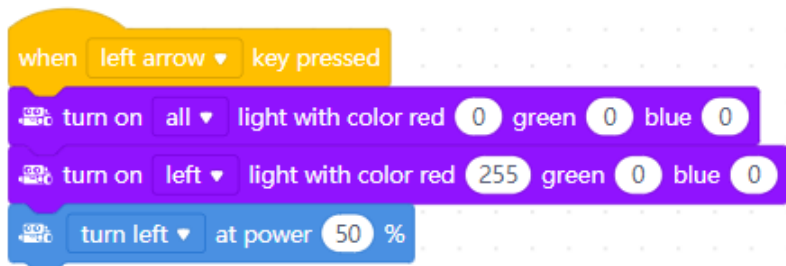
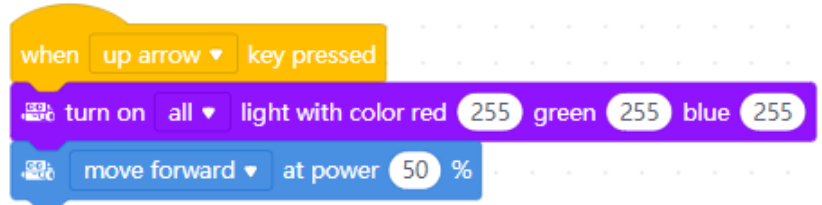
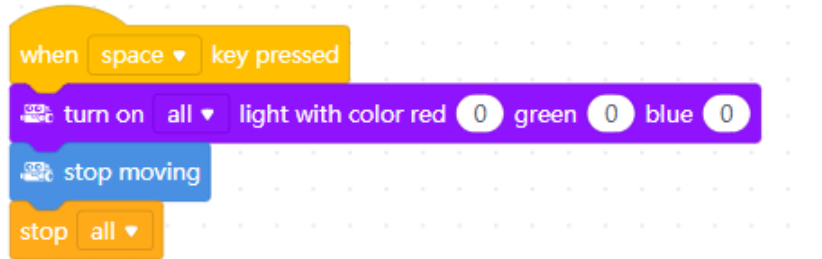
The **'Pause_Timer'** block shown in my **'mBot start'** script (on the previous page) is called several times in the other block scripts in this project. This block can be easily edited to change the contents and is currently set in my programme to **'wait (0.25) seconds'**.



It is easily changeable and is added to briefly pause script sequences to give mBot

(mCore) time to process or store data.

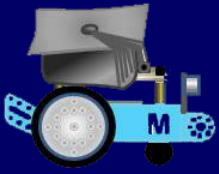
There is also a second timer block, **'Move_Timer'** which is called once only. This sets the length of time that mBot moves forwards before stopping to poll the sensor. I have set this to **'wait (1) second'**. This block is the same as the very last block (**'wait (0.6) seconds'**) that can be seen at the bottom of both of the original script and the transcribed version shown on page 207.



The two remaining self-defined block scripts (**'Poll_Range'** and **'Make_Decision'**) and called in the **'forever'** loop part of the **'mBot start'** script, are all shown on the next page.

The self-defined **'Poll_Range'** block checks and then stores the feedback from the sensor at three different (**'Range_Left'**, **'Range_Centre'** and **'Range_Right'**) head positions.

The **'Make_Decision'** block calls several other self-defined blocks **'Move_Forwards'**, **'Move_Backwards'**, **'Turn_Left'**, **'Turn_Right'** and **'Head_Feedback'**. These last five blocks break down the decision-making process about 'how-and-when' to turn into even more understandable chunks.



mBot and Me a Beginner's Guide

define Poll_Range

set Range_Left to round ultrasonic sensor port3 distance(cm)

servo port4 slot2 positioned at 90

Pause_Timer

set Range_Centre to round ultrasonic sensor port3 distance(cm)

servo port4 slot2 positioned at 40

Pause_Timer

set Range_Right to round ultrasonic sensor port3 distance(cm)

servo port4 slot2 positioned at 90

Pause_Timer

define Make_Decision

if Range_Centre < 25 then

Move_Backwards

Pause_Timer

if Range_Right - Range_Left > 0 then

Turn_Right

else

Turn_Left

else

Head_Feedback

Move_Timer

As discussed on the previous page, here are the final seven scripts for my 'Head-Shaking Cat' (mk. I) model).

define Move_Forwards

turn on all light with color red 255 green 255 blue 255

move forward at power 50 %

define Move_Backwards

turn on all light with color red 0 green 0 blue 255

move backward at power 50 %

define Turn_Left

turn on all light with color red 0 green 0 blue 0

turn on left light with color red 255 green 0 blue 0

turn left at power 50 %

define Turn_Right

turn on all light with color red 0 green 0 blue 0

turn on right light with color red 0 green 255 blue 0

turn right at power 50 %

define Head_Feedback

if Range_Left < 15 and Range_Right - Range_Left > 0 then

Turn_Right

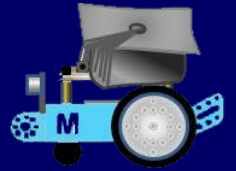
else

if Range_Right < 15 and Range_Left - Range_Right > 0 then

Turn_Left

else

Move_Forwards



Programming the 'Head-Shaking Cat' (mk. II) model

Once I had created the 'tidied-up' version of the original 'The Kitten Looking Around' Makeblock project file (as described in Appendix 2 on page 198 and shown on page 207) I thought that I might be able to improve on the rather irksome 'move, pause, look-around, move again etc.' sequence of the original project and I had my own ideas about how I might create a viable alternative.

When I first worked on my version of this project (initially using mBlock 3) it made considerable sense to use the versatile 'Events' > 'when (space) key pressed' / 'released' hat blocks to control the programme so that mBot could be stopped easily if it got into trouble. But the 'when (space) key released' block now no longer exists! Now converted to mBlock 5 format, my programming scripts mostly work as I expected and are fairly understandable sequences.

I did however find that the ultrasonic sensor feedback into 'Proximity' was occasionally not good when mBot was heading at a shallow approach angle towards a wall or other major obstructions. Whilst this programme did work, I found that it was not quite as reliable as the transcribed original 'The Kitten Looking Around' Makeblock file (described in detail on pages 211 to 215).

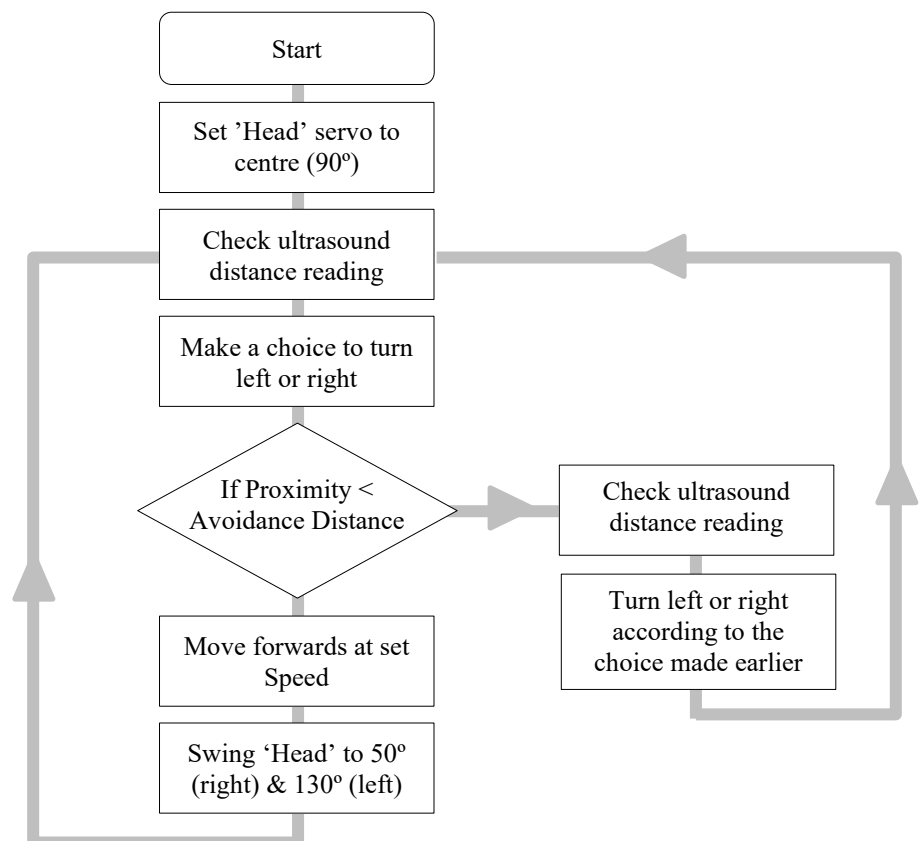
My own Project Concept for the 'Head-Shaking Cat' (mk. II) - in mBlock 3 was:

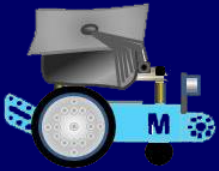
The 'Cat' moves for as long as the SPACEBAR on the keyboard is depressed. It moves forwards shaking its 'head' from side-to-side until it detects something close to it – the 'cat' then chooses to turn randomly to either to the left or to the right and only stops if the SPACEBAR is released.

BUT now I had to rethink this for mBlock 5. ***If I can overcome the start / stop problem then this concept is still viable!***

Shown on the right is my original Flow Chart algorithm which describes the logical processes required to achieve the objectives of the concept.

Since mBot needs to move and swing it's 'head' it needs to be able to multi-task, so a 'Proximity' check needs to be done frequently (and just before it's needed) to make the decision whether 'to turn or not to turn'. I needed a variable 'Set_Proximity' to store feedback (in millimetres) from the ultrasonic distance sensor.





mBot and Me a Beginner's Guide

I also decided that I needed a self-defined block function 'Set_Turn' which would use a random number (0 or 1) and turn this numeric decision into understandable text, "LEFT" or "RIGHT", and then store the textual direction to turn in a variable called 'Random_Choice'.

I also needed a second block function 'Make_Turn' to make the decision when to turn, and which way to turn, using the following piece of logic:

**If 'Proximity' is less than 'Avoid_Distance' and 'Random_Choice' contains "Left" then turn LEFT
else**

If 'Proximity' is less than 'Avoid_Distance' and 'Random_Choice' contains "Right" then turn RIGHT

To start programming this model you first need to make three descriptively named Variables 'Avoid_Distance', 'Proximity' and 'Random_Choice'. The use of these names will help to make your scripts clearly understandable. You also need to make four self-defined 'My Blocks' - 'Make_Turn', 'Move_Sensor', 'Set_Proximity' and 'Set_Turn'. Not only will they break the programming concept into small, easily understood chunks, but these names will also help to make the whole programming sequence clear.

Eventually I decided that the best way to start and stop the 'Head-Shaking Cat' was by using two separate keypresses. Not as good as holding and then releasing a single key, but as a workaround this solved the loss of the 'when space key released' hat block.

On the right is the 'mBot start' script I created to activate the 'Head-Shaking Cat'.

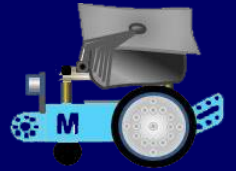
This is initiated by pressing the **Zero (0) key** on the keyboard and starts by repositioning the 'head' (the Ultrasonic Sensor) centrally at 90°.

For most keyboards, the numeric keypad usually has a 'Zero' (0) key at the bottom of the keyboard just to the right of the cursor keys and on the same line as the 'Spacebar', so this is also a good choice to start mBot.

This sequence calls the 'Set_Proximity' block script to take feedback (in millimetres) from the ultrasonic distance sensor, generating a value in a variable called 'Proximity'. The 'Set_Turn' block script called next generates a random number to put a textual value 'LEFT' or 'RIGHT' into another variable called 'Random_Choice'.

The 'Make_Turn' block script (shown on the next page) uses the 'if ... then ... else' decision described at the top of this page. It is also much easier to understand since it uses the text values 'LEFT' or 'RIGHT' as decision choices. It is nested inside the 'forever' loop (shown above) which also moves the 'cat' forwards and moves its 'head' from side to side.





I added the same *'mBot stop'* script (activated by the spacebar) and the four manual control scripts (using the cursor keys on a keyboard) that I had created for my *'Head-Shaking Cat'* (mk. I) project (see page 209). The 'up arrow' cursor key is *'mBot forward'* whilst the 'down arrow' cursor key is *'mBot reverse'*, the cursor key 'left arrow' is *'mBot left'* and the cursor key 'right arrow' is *'mBot right'*.

Shown below are the four self-defined function blocks called by the main *'when (0) key pressed'* hat block script (described and shown on the previous page).

```

define Set_Turn
if pick random 0 to 1 = 0 then
set Random_Choice to LEFT
else
set Random_Choice to RIGHT

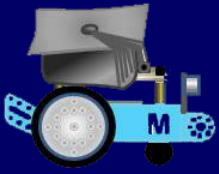
define Make_Turn
if Proximity < Avoid_Distance and Random_Choice = LEFT then
turn left at power 50 % for 0.5 secs
else
if Proximity < Avoid_Distance and Random_Choice = RIGHT then
turn right at power 50 % for 0.5 secs

define Set_Proximity
set Proximity to round ultrasonic sensor port3 distance(cm) * 10

define Move_Sensor
turn on all light with color red 0 green 0 blue 0
servo port4 slot2 positioned at 50
turn on all light with color red 0 green 255 blue 0
wait 0.25 seconds
Set_Proximity
turn on all light with color red 0 green 0 blue 0
servo port4 slot2 positioned at 130
turn on all light with color red 255 green 0 blue 0
wait 0.25 seconds
Set_Proximity
    
```

I found that if my mBot 'cat' was trapped in a corner or heading at a shallow approach angle towards a wall or other major obstruction then I needed to quickly hit the *'Spacebar'* to activate my *'mBot stop'* script which stops and resets everything!

I was then able to turn or reverse mBot out into open space once more by using one or more of the four cursor key manual control scripts.



mBot and Me *a Beginner's Guide*

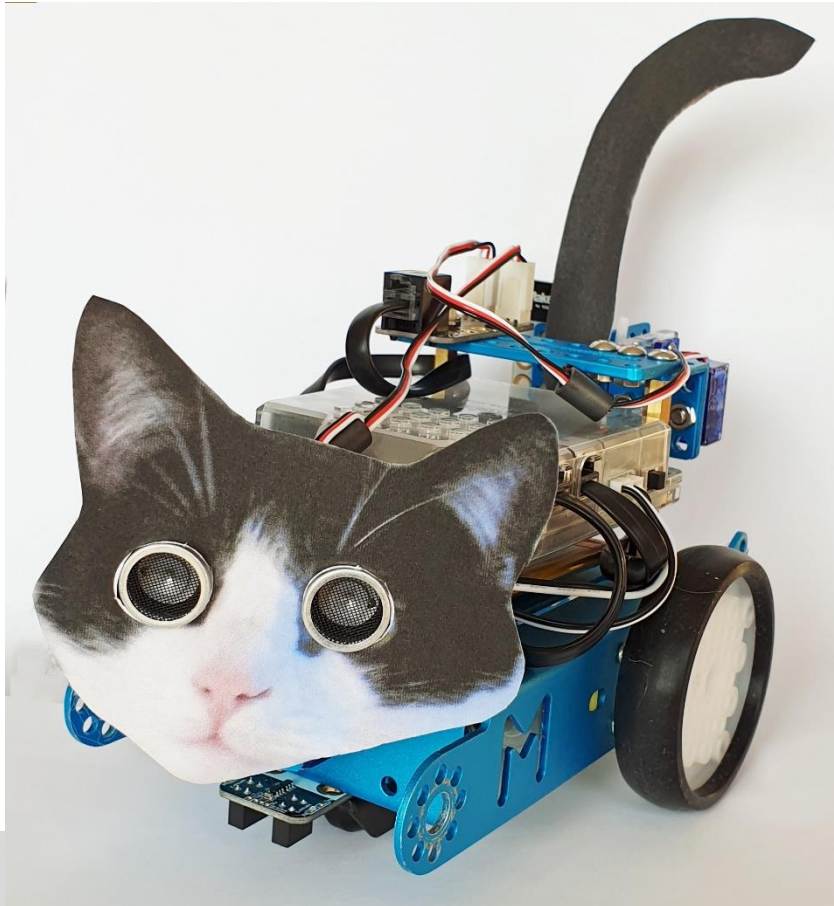
Whilst this programme worked, I found that it was not as good as the original Makeblock project file '*The Kitten Looking Around*' which had the hesitant stop-and-look-approach (see pages 206 to 210).

You can see from the picture on the right that for both the (*mk. I*) and the (*mk. II*) models I modified mBot's 'eyes' (the ultrasonic sensor based moving 'head') by adding a cat mask over it.

Emma's cat 'Ko-Ko' once again provided the mask and Emma, whilst she did not think that this project was as good as '*The Dancing Cat*', loved watching the head move from side to side.

N.B. You can also see the original servo mount in place at the back of the model and a marginal modification to the second servo (tail-raising) option too.

I didn't actually do it, but I considered adding the tail raising scripts from my

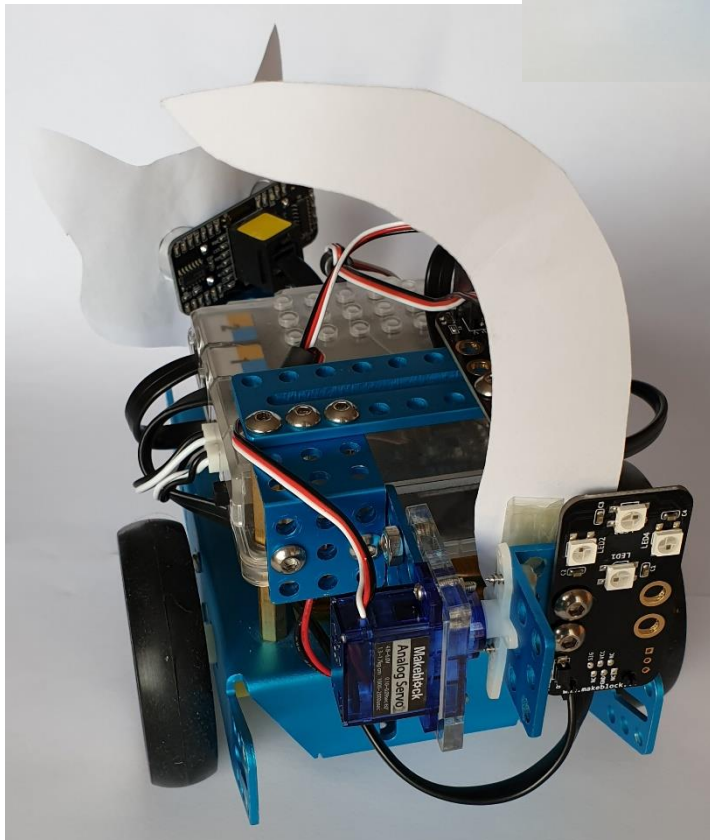


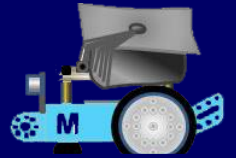
'*Dancing Cat*' project here, setting it to raise the tail when moving forwards and lowering it when mBot stopped.

This would have been fairly easy to implement, but I didn't want to add any more to the content of this project unnecessarily - if you have a second servo then you might want to try adding the required scripts here yourself.

What I actually did do, as you can see in the picture on the left was to modify the rear servo mount and add an RGB-LED module to a 3x3 angle bracket mounted on the servo arm in preparation for the next project (the '*Light-Emitting Cat*' model).

This modification is discussed in Appendix 5 which begins on the next page.

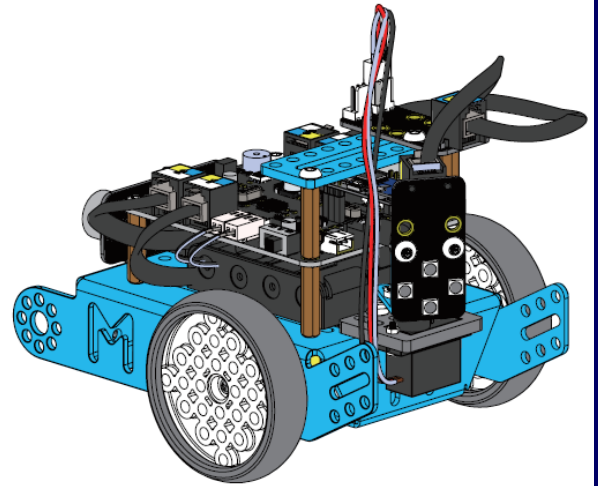




Appendix 5 - mBot Servo Project - 'Light-Emitting Cat'

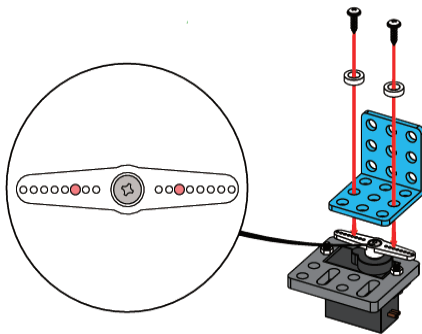
The modifications to that need to be made to mBot to create the 'Light-Emitting Cat' model (as described in the construction notes in the Servo Pack .pdf file) are essentially simple and rather uninspiring.

The servo is just repositioned by moving it from the front of mBot to the rear and attaching the Me RGB-LED module to it to represent the cat's tail. (See the illustration on the right).

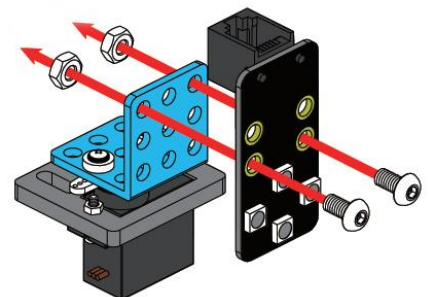


The instructions suggest that you need to leave the RJ25 Adapter Module attached to the plate on top (at the back) of mBot, exactly as I have suggested for my 'Dancing Cat' model and both of my 'Head-Shaking Cat' models.

Once again, I left the protective cover in place over the top of mCore. The 9g Micro Servo should still be on its mounting plate - but this time it needs the servo arm to be repositioned horizontally (as it was for the 'Dancing Cat' model) - not vertically as for the 'Head-Shaking Cat' model.



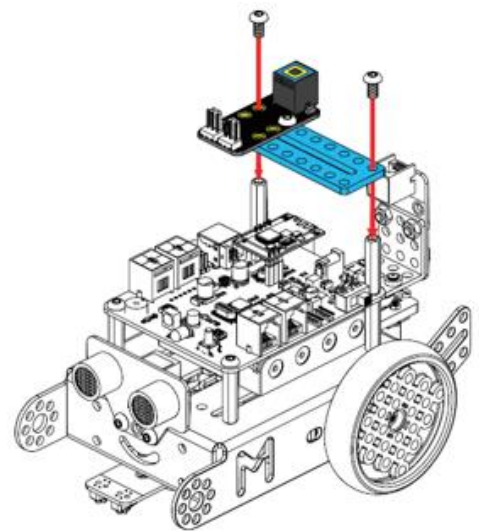
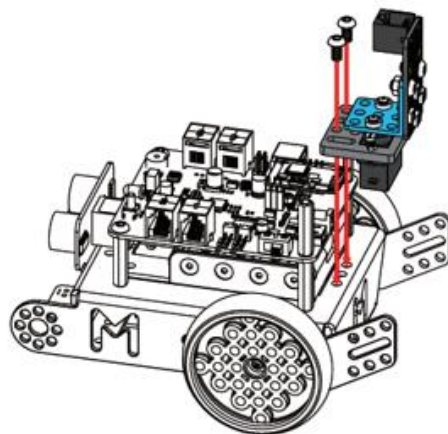
The 3x3 angle bracket is then mounted on to the servo arm as shown in the diagram on the left. Next, attach the RGB-LED module to the 3x3 angle bracket using two screws and nuts as shown in the diagram on the right.



Mount this complete servo assembly on to the rear of mBots chassis as shown in the diagrams below.

It is fairly fiddly to hold the servo assembly in place whilst attaching nuts to the two locating screws on the underside of the chassis.

You may find this easier to do with the 9-hole x 2 Slotted Plate and Me RJ25 Adapter Module removed (as shown on the right).



I found that holding the screws with small snipe-nosed pliers helped here.



mBot and Me

a Beginner's Guide

Reconnect all the Me components to mCore. You can use other ports for this project if you wish, however it makes sense to leave the infrared emitting & receiving LED line-following module connected to port2 and the ultrasonic sensor connected to port3 (exactly as they were in the original mBot configuration). The Me RJ25 Adapter module should once again be connected to port4 and the servo connected to slot2 of the adapter board. That leaves port1 available for connecting the RGB-LED module (the cat's flashing tail) using another 6P6C RJ25 Me cable.

The 'Light-Emitting Cat' mBlock 3 (.sb2) file downloaded from Makeblock and described in Appendix 2 on page 198, was also not very inspiring (see the complete set of original scripts for this below-left).

```
when space key pressed
  set servo Port4 Slot2 angle 90
  forever
    Set_Proximity
    Flash_LEDs
    Setup_Turn
    set Feedback 2 to join Next Random Turn = Random
    if Proximity > Avoid_Distance then
      run forward at speed Speed
      Move_Tail
    else
      Set_Proximity
      Flash_LEDs
      Make_Turn

define Set_Proximity
  set Proximity to round ultrasonicsensor Port3 distance * 10
  set Feedback 1 to join Ultrasonic Sensor Reading Proximity

define Make_Turn
  if Proximity < Avoid_Distance and Random = LEFT then
    turn left at speed Speed
    wait 0.5 secs
  else
    if Proximity < Avoid_Distance and Random = RIGHT then
      turn right at speed Speed
      wait 0.5 secs

define Flash_LEDs
  set led on board led left red pickrandom 0 to 40 green pickrandom 0 to 40 blue pickrandom 0 to 40
  set led on board led right red pickrandom 0 to 40 green pickrandom 0 to 40 blue pickrandom 0 to 40
  set led Port1 1 red pickrandom 0 to 40 green pickrandom 0 to 40 blue pickrandom 0 to 40
  set led Port1 2 red pickrandom 0 to 40 green pickrandom 0 to 40 blue pickrandom 0 to 40
  set led Port1 3 red pickrandom 0 to 40 green pickrandom 0 to 40 blue pickrandom 0 to 40
  set led Port1 4 red pickrandom 0 to 40 green pickrandom 0 to 40 blue pickrandom 0 to 40

when space key released
  run forward at speed 0
  set servo Port4 Slot2 angle 90
  set led on board all red 0 green 0 blue 0
  stop all

define Setup_Turn
  if pickrandom 0 to 1 = 0 then
    set Random to LEFT
  else
    set Random to RIGHT

define Move_Tail
  Flash_LEDs
  set servo Port4 Slot2 angle 60
  wait 0.25 secs
  Set_Proximity
  set servo Port4 Slot2 angle 120
  wait 0.25 secs
  Set_Proximity

when clicked
  set Avoid_Distance to 400

when clicked
  set Speed to 200

when clicked
  Setup_Turn

when clicked
  set led Port1 all red 0 green 0 blue 0
  set led on board all red 0 green 0 blue 0
```

Having played with this programme in mBlock 3, I didn't think it worth it to transcribe any of these scripts into mBlock 5 format.

As hinted at on page 214, I thought that it would be much better if could use the scripts created for both moving mBot and turning the head in the 'Head-Shaking Cat' mk. I project in Appendix 4.

All I needed to do, I thought, was to duplicate this file and add a couple of new self-defined blocks.

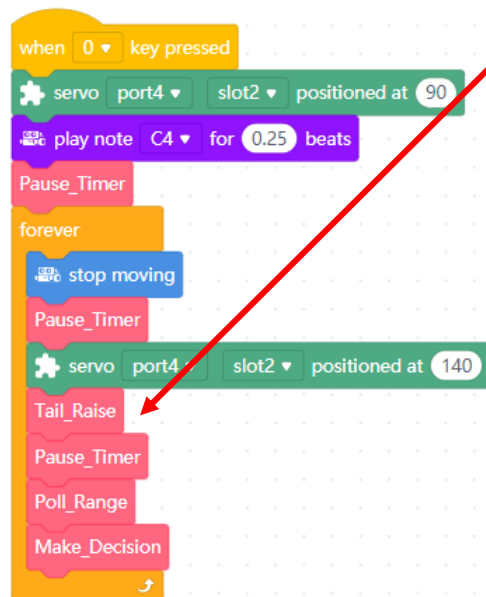
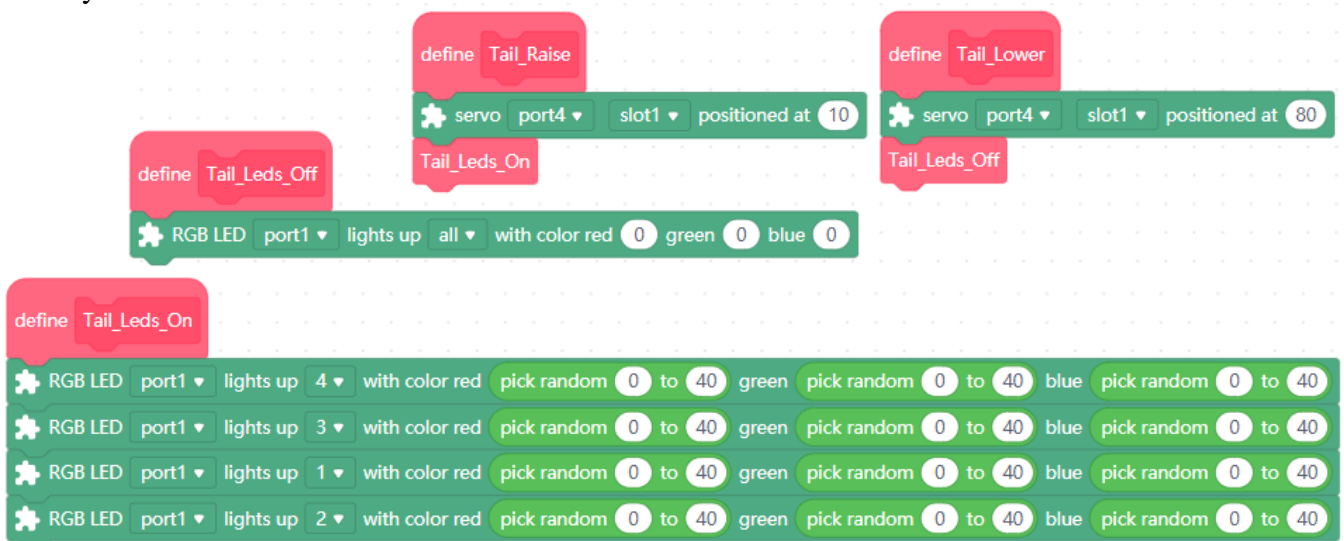
Particularly a script to randomly flash the Me RGB-LED module lights.



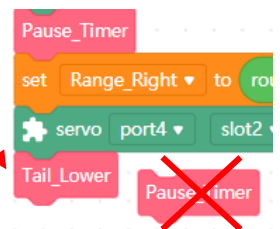
I would also need to add very similar tail-raising and tail-lowering blocks to those created in my 'Dancing Cat' project. So, I loaded my 'Head-Shaking Cat' project file into mBlock 5 and immediately saved it as a new project file called 'Light-Emitting Cat'; but before continuing with any programming I had to reconfigure mBot.

If you are moving on from the 'Head-Shaking Cat' model described in Appendix 4 (and I assume that you are) then there is very little to do in terms of any modifications needed to turn mBot into the new model.

I only needed to add four new 'My Blocks' to the new 'Light-Emitting Cat' project. These self-defined blocks were called 'Tail_Raise', 'Tail_Lower', 'Tail_Leds_On' and 'Tail_Leds_Off' and are shown in entirety below:



I added the new block 'Tail_Raise' here in my 'mBot start' script and then added the 'Tail_Lower' script to the very end of the 'Poll_Range' script replacing the 'Pause_Timer' block that was at the end of the script - no need to wait for mCore to process data on this occasion since that could happen whilst the servo was being operated.



The four little ws2812 lights on the RGB-LED module are set to show random colours each time that the 'Tail_Leds_On' script is called.

My script sequence illuminates the individual ws2812 LEDs in a clockwise direction from the top (position 4 - 12 o'clock) to the left-hand (position 2 - 9 o'clock).

The 'Tail_Leds_Off' script conversely sets the three numeric permutations (for all four of the little ws2812 LED lights colours) to zero - which means no light is emitted at all.



Addendum re 'Cats'

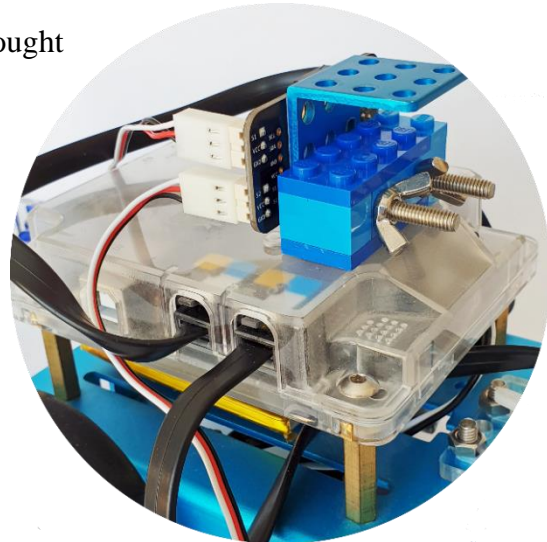
By now, I was getting slightly fed-up with 'Cat' models - but I *had* completed the last one and had ended up with four complete 'Cat' projects. After the confusion over their names (see page 198) it now seemed to make sense (to me anyway) to totally rename them as follows:

- 'Dancing Cat' - this project I renamed as **'Happy Cat'**
- 'Head-Shaking Cat mk. I' - this project I renamed as **'Cautious Cat'**
- 'Head-Shaking Cat mk. II' - this project I renamed as **'Inquisitive Cat'**
- 'Light-Emitting Cat' - this project I renamed as **'Colourful Cat'**

Re. the Me RGB-LED module used in the 'Light-Emitting Cat' / 'Colourful Cat' project; Appendix 11 on page 233 discusses the basic principles of testing and experimenting with LEDs in detail.

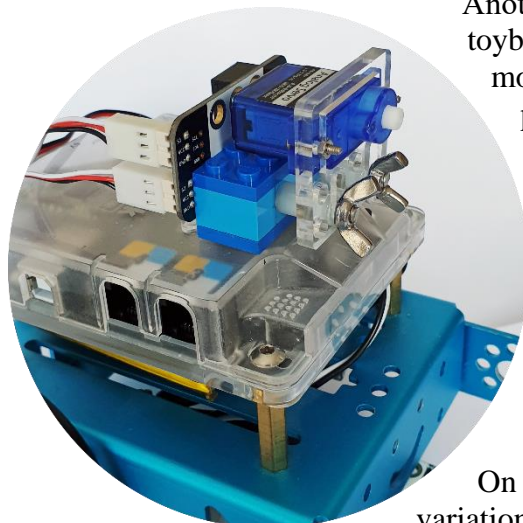
Towards the end of my 'experiments with cats!' I suddenly thought about simplifying the models even more. Most of the construction modifications to mBot for all of the models involved mounting the RJ25 Adaptor board onto the 9-hole (I1) Slotted Plate which was supported on two brass extender studs screwed into the rear on mBot's case.

This was certainly needed for the 'Dancing Cat' model - but not for the other models; so what if this rather unsightly assembly was removed altogether and the RJ25 Adaptor mounted on another 'Lego' block that could be positioned anywhere on the studs at the front end of the transparent case covering the mCore board.

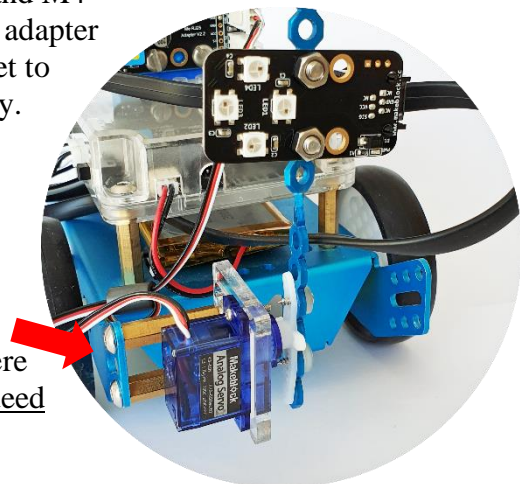


Another trip to Emma's toybox was needed! The resultant mount (exactly the same as the one described in Appendix 1 on page 184) is shown above-right.

This block has proved to be so versatile, and you can see in the image above that I also added a 9-hole angle-bracket to the assembly to allow a variety of other mounting options. I used two M4 x 35 Screws and M4 wingnuts to attach the adaptor board and angle-bracket to the Lego block assembly.



On the left is another variation, this time adding a servo (for the 'cat's waving paws' perhaps) to the block. Finally, in the image on the right you can see the very simple modification I made to mount a second servo 'cat's tail' unit to the rear of mBot using two brass extender studs. You can see here (compared to both of the pictures on page 214) that these are indeed much neater modifications.





Appendix 6 - mBot 'add-on' component - Robot Pack

Makeblock Model No. 98050 - the Six-Legged Robot Pack

The mBot Six-Legged Robot pack is another add-on 3-models-in-1 pack for mBot. The MakeBlock advertising material for this pack says that you can construct either a *'Beetle'*, a *'Mantis'* or a *'Crazy Frog'* with the components in the pack and suggests that these additions make the robot more enjoyable and benefit children's creative thinking. Information in this pack (see below) is, as usual, fairly minimal - bought in the UK it cost me £18 plus a marginally steep £3.50 p&p (but it did arrive the next day!).

The advertising blurb also has flowery descriptions - the primary model, a 'walking hexapod', is described as "*a six-legged Beetle that can turn around quickly to attack an enemy*" and a second model "*The Mantis quietly crawls, waving two arms and is a hunter from the darkness*". It also describes the third model as "*the little crazy Frog which dashes around so madly that nobody can stop it*".

The little manual that comes with the pack shows a sequence of graphics similar in style to the original mBot manual; these images only show how to modify mBot to create the so-called 'Beetle' and construction details of the other two models are not included. Much more importantly, and as disconcertingly as before, there is no hint of how to programme any of the models using mBlock.

But now that I knew where to look - thanks to searching previously for the elusive Servo Pack Instructions (see pages 196 & 197). I found the complete model making instructions and some programming examples on the Makeblock page describing add-on packs that I had visited earlier using the following link:

<http://learn.makeblock.com/en/mbot-add-on-packs/>

The programming files were once again stored as RAR files, so as before, I used 7-Zip to decompress them. On examining these mBlock files I was delighted to see that everything was in English including the comment callouts attached to some of the blocks in the scripts - they *were* written in English, but (as before) needed some thought to clarify the meaning!

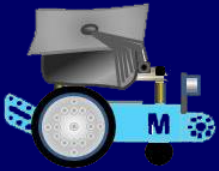
The enclosing box was the usual size, shape and style for these add-on packs and on first-opening, was a little underwhelming since it contains no new Me Modules (but I was expecting that!). This add-on pack once again contained the useful Mini-Spanner (Wrench) - so I now have 3 of these!

In the box too there was a huge bag of assorted (and very useful fasteners) but the most exciting and useful contents were undoubtedly the 12 shiny anodised blue beams and plates which matched my own mBot. I now had at my disposal enough constructional bits to experiment with my own models, so buying this add-on pack is certainly worth it for the contents alone and not particularly for building the models specified.

In detail, the beams and plates contained in the pack are as follows:

Two 11-hole, **108mm 0412** Beams (see the next page for the **blue/red** coding here).

There are two 10-hole, 92mm 0412 Beams; six 9-hole, 76mm 0412 Beams and two 7-hole, 60mm 0412 Beams. There are also two seven-hole 45° Plates - you may have gained one of these already, if like me, you have bought (or will buy) the 'Light & Sound' Add-On Pack.



An example of Makeblock's description method:

Beam **0412-108** ...

... refers to a beam's cross-sectional dimension = **4mm x 12mm**

... and its length = **108mm**



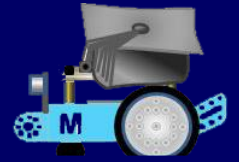
Every component in the 'Six-Legged Robot' add-on pack is shown laid out in the picture above.

Below is a list of the many fasteners you get in the pack. Do note that once again there *are* several more of these than actually specified by Makeblock!

- 10 × M4 x 8 Screws
- 20 × M4 x 14 Screws
- 10 × M4 x 22 Screws
- 4 × M4 x 30 Screws
- 4 × M4 x 35 Screws
- 10 × M4 Nuts
- 20 × M4 Nylock Nuts
- 20 × 3mm x 7mm dia. Plastic Spacers
- 12 × 10mm x 7mm dia. Plastic Spacers
- 24 x Plastic Peg Rivets (4mm dia. x 15)

N.B. The Plastic Peg Rivets together with 3mm thick spacer washers can be used in lieu of screws and nuts as movement pivots to speed up model building & experimentation with your own projects (but be advised that for model reliability, using screws with lock nuts is a much better option).

In truth I was looking forward to this add-on pack very much. Especially gaining the Beams to use in my own creations. I thought that 'walking' robot models might be good value too and entertaining though they were, they were also rather boring after a few goes with the IR remote control. Sadly, there is not really any scope for mBlock Scratch programming with these models other than Emma's wishful idea for using the sound sensor to get the 'Frog' to jump on command - something we decided that was just not worth it; so (and at variance with the advertising blurb) there was very little here to inspire Emma into any really creative thinking.

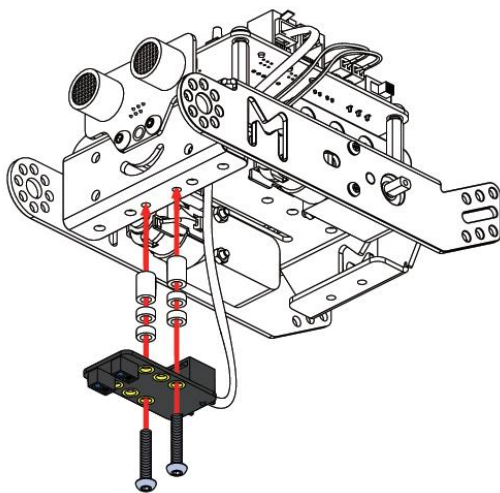


Appendix 7 - mBot Walker Project - Robot 'Beetle'

Building / modifying the 'Six-Legged Beetle' model

The 'Six-Legged Beetle' mBot variant is quite a complex (but not too difficult) construction; and once again, some aspects of the model are fiddly and a little awkward for big fingers.

The principle of this model (a 'walking hexapod') relies on inserting Screws into the main drive wheels of mBot to serve as 'cranks' to operate a set of levers. Each 'crank-pin' is attached off-centre to each of the wheels and this acts like a cam producing piston-like linear motion as each wheel rotates - which then drives a linkage of several other levers in a series of crude step-like movements. This is the opposite of your legs pushing on the pedal cranks of a bike to generate rotary motion!



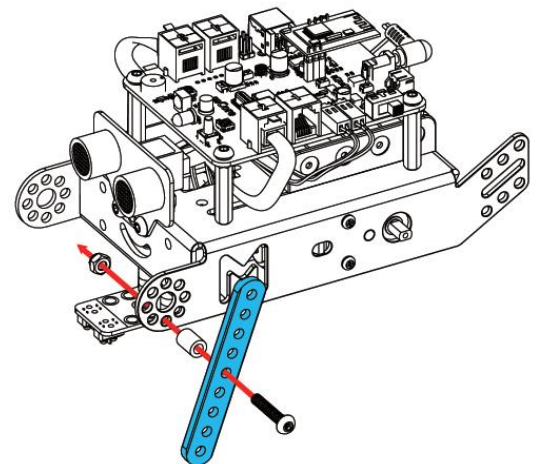
If you are starting with a completely built mBot then you will need to disassemble the front wheel and Me Line-follower Sensor assembly. Store the front wheel unit somewhere safe for rebuilding mBot later. Remove the two wheels and remove their tyres - store the tyres somewhere safe too (& do not lose the two little self-tapping screws).

Next, re-fit the Line-Follower Sensor 16mm below the underside on mBot's chassis using two sets of spacers (one 10mm & two 3mm over each screw) and using two M4 x 22 Screws as shown in the diagram on the left.

Note which holes in the Line-Follower Sensor Module and on the chassis are used for this.

Assemble the left-side leg linkage of the 'Beetle' first. Start by passing one M4×22mm Screw through the centre hole of a 9-hole, 76mm 0412 Beam and then through a 7mm dia. × 10mm long plastic spacer. Fit the screw through the bottom hole of the circular pattern of eight holes at the front of mBot's chassis and secure it in place with an M4 Lock Nut, allowing the Beam to move freely and smoothly (see the diagram on the right).

N.B. If the joint is too loose, the leg might jam, and mBot will not be able to walk easily; but if the joint is too tight, then mBot won't be able to walk at all!

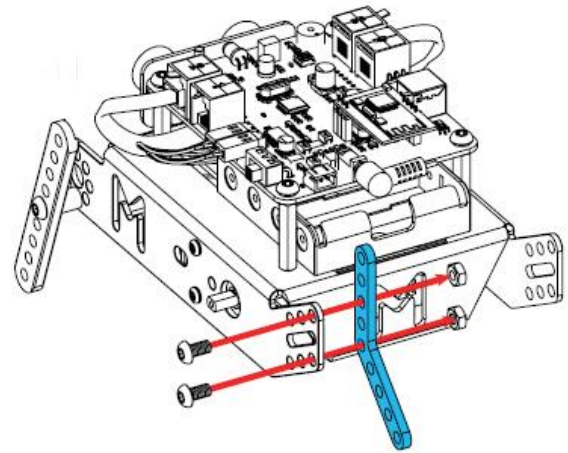




mBot and Me *a Beginner's Guide*

Next, using two M4×8mm Screws and two M4 Nuts, attach a 9-hole 45° Plate to the rear of mBot's chassis as shown in the diagram on the right. (Remember, this is still the leg linkage assembly on the left-side of mBot's chassis).

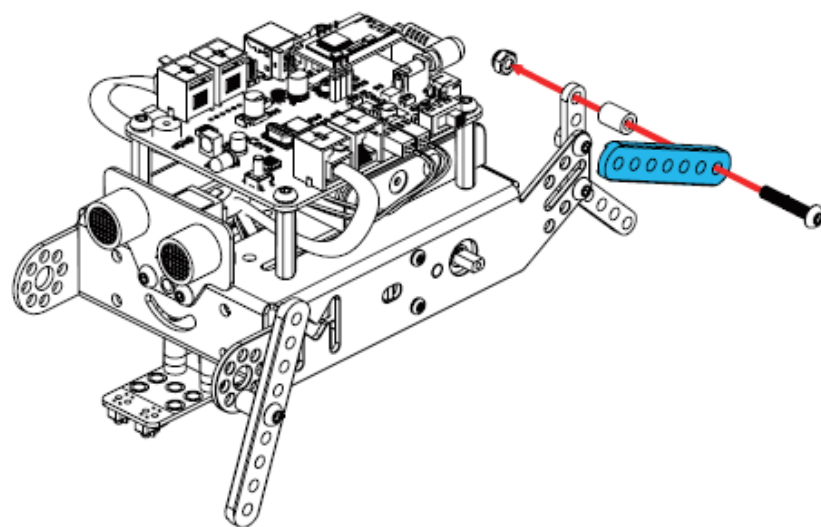
Push an M4×22mm Screw through the end hole of a 7-hole, 60mm 0412 Beam. Slide on to the screw a 7mm dia. × 10mm long plastic spacer and fit this assembly through the top hole of the 9-hole 45° Plate you added to the rear of mBot in the last step.



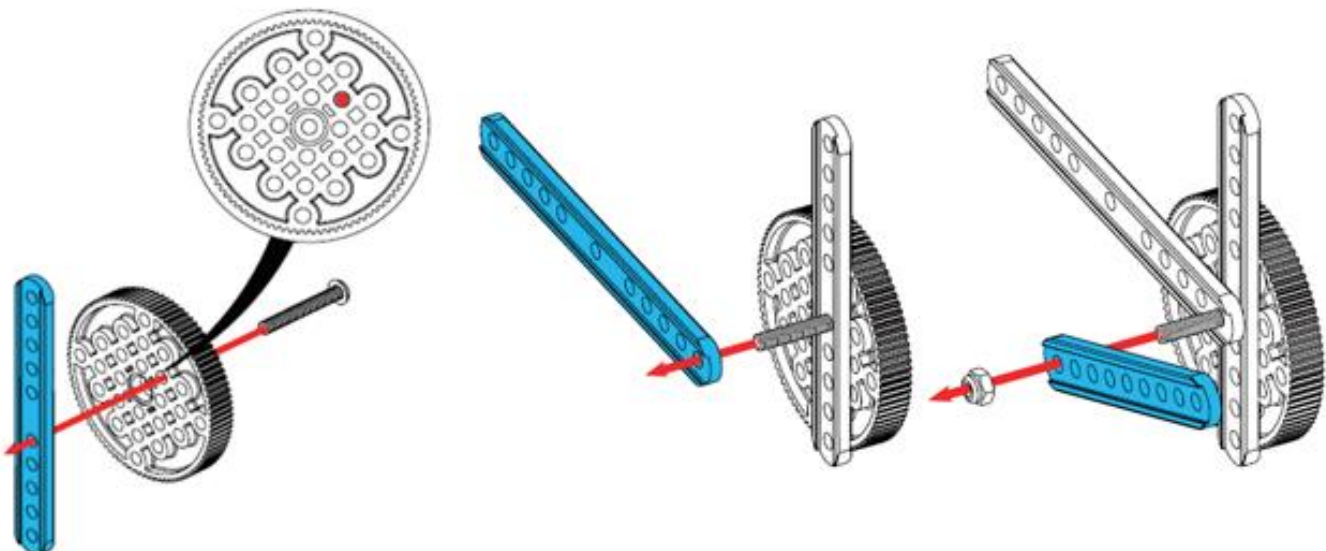
See the diagram on the left.

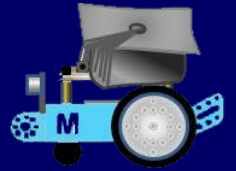
Secure the assembly in place with an M4 Lock Nut, allowing the Beam to just move freely - once again not too tight & not too loose either!

Below is shown one of the main drive wheels. Insert (from the back of the wheel) an M4×30mm Screw through the hole marked red and slide over it a 10-hole, 92mm 0412 Beam using the fifth hole in from one end.



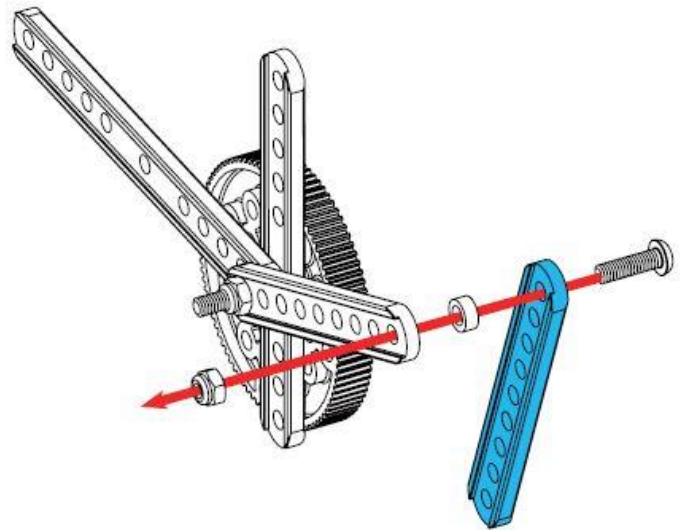
On top of that, place an 11-hole, 108mm 0412 Beam, passing the screw through the end hole as shown in the sequence below. On top of those two Beams add a 9-hole, 76mm 0412 Beam using an end hole once again. You can now use an M4 Lock Nut to secure these three Beams to the Wheel - remember, not too tight & not too loose!





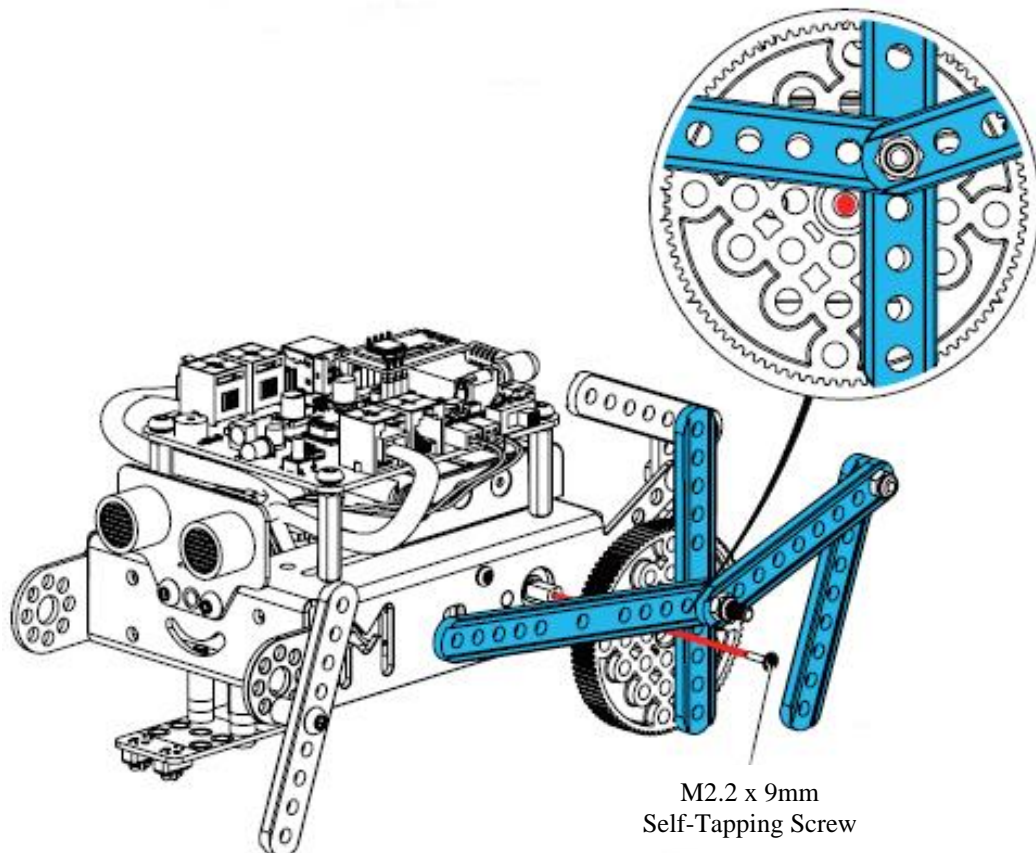
Into the end hole of the 9-hole, 76mm 0412 Beam you have just added, add an M4×14mm Screw passing through another 9-hole, 76mm 0412 Beam using the end hole once again as shown in the diagram on the right.

Make sure that you put this new Beam behind the original, separating them with one 7mm dia. × 3mm thick Plastic Spacer. Secure this new Beam in place with an M4 Lock Nut, allowing once again the Beam to move smoothly and freely - not too loose and not too tight.



Now it's time to replace the left-side drive wheel back on to its axle using one of the two M2.2 x 9mm Self-Tapping Screws (that you put aside when you removed the wheels earlier) to secure it firmly on to the shaped axle sticking out of the motor and through mBots chassis.

Your wheel assembly and the leg linkages that you have created so far for the left-hand side of mBot should look like the diagram below:



M2.2 x 9mm
Self-Tapping Screw

Do remember for all of your Linkage joints - make them not too tight & not too loose!



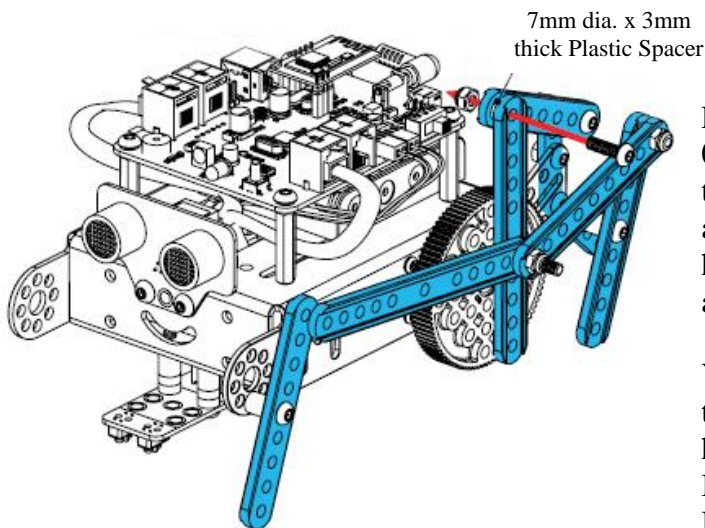
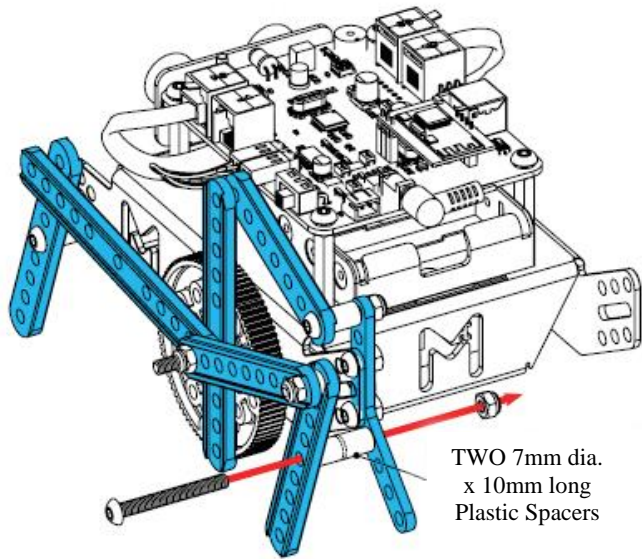
mBot and Me a Beginner's Guide

Use an M4×35mm Screw passing through the fifth hole in the rear leg of the 'Beetle' (as shown in the diagram on the right) using two 7mm dia.×10mm long Plastic Spacers to separate the two Beams.

Pass the Screw through the fourth hole up in the 9-hole 45° Plate you attached to the rear of mBot's chassis earlier and secure it in place with an M4 Lock Nut, checking that both Beams move smoothly and freely; once again, not too loose and not too tight.

It's finally time to join-together the linkages for the left-side of the 'Beetle'.

There are just two bits to link together.



First, you need to go back to the 7-hole, 60mm 0412 Beam that you attached with a plastic spacer to the top hole of the 9-hole 45° Plate that you added to the rear of mBot earlier (you left this hanging freely whilst you dealt with the wheel assembly). See the diagram on the left.

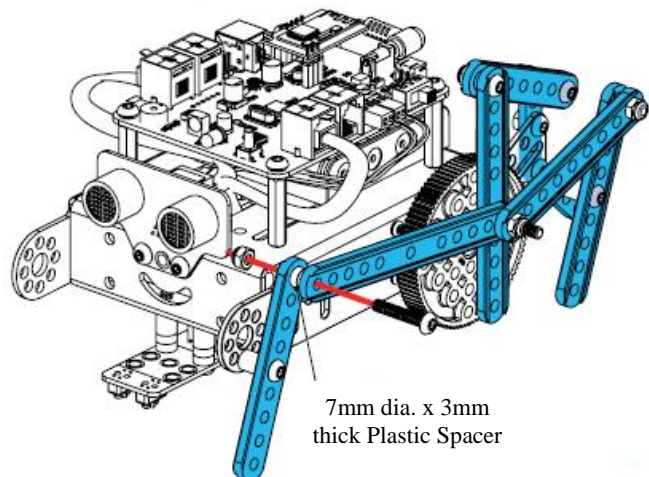
You need to attach the last hole of the free-end of this Beam (the free-end where you can see 5 holes) to the end hole of the 10-hole, 92mm 0412 Beam that you added first to the wheel, using an M4×14mm Screw passing through a 7mm dia. x

3mm thick Plastic Spacer and secured in the usual way with an M4 Lock-Nut.

Finally, you need to connect the very first Beam that you attached to the front of mBot's chassis (which has also been hanging free since the start of construction) to the end hole in the free-end of the long 11-hole, 108mm 0412 Beam.

Use an M4×22mm Screw passing through a 7mm dia. x 3mm thick Plastic Spacer between the two Beams to make this connection and secure it in the usual way with an M4 Lock-Nut as shown in the diagram on the right.

The left-side linkages that comprise one half of the 'Beetle's' legs are now complete.





It is not too difficult a task to repeat a mirror-image construction of the left-side linkage to create the linkage for the right-side legs of the 'Beetle' - but do note the comment about the 'crank' Screw position below. You should find this second side to be a much quicker assembly time. In reality, I mirrored the construction as I added each component - first the left-side component and then the matching right-side one.

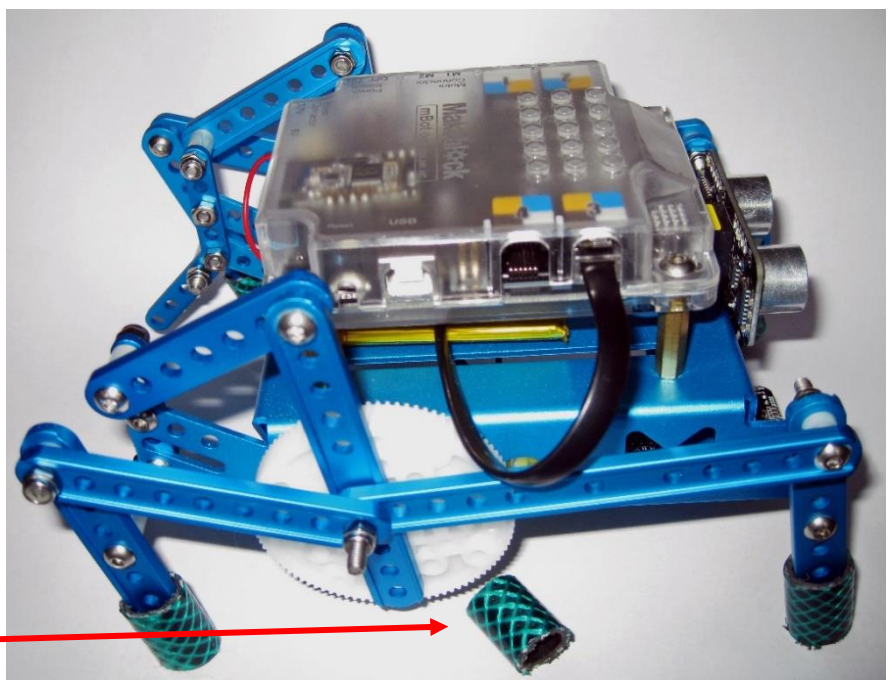
*N.B. Once I had made up the wheel for the left-hand side of the 'Beetle' I copied its construction to match the position of the three linkage Beams to be added to the right-side wheel **but making sure that the Screw providing the 'crank' was in a mirror image position** (i.e. it was on the left side of the central axle hole not on the right of centre like the red dot in the diagram illustrating the left-side).*

This all worked very well, but it probably took about an hour altogether to construct this model. This has been the longest construction time for any add-on model so far - but was it worth it? Remember, it's important to find the balance between loose & tight with the linkages and perhaps check the joints every time you operate it. Remember too that if the joints are too loose the legs will jam and mBot will not walk. If the joints are conversely too tight then mBot can't walk at all. I used the principle of screwing them tight till they were solid and then unscrewing them for half-a-turn, which seemed to work well.

Was it worth it? Yes, I think that it was - *just* - if only for novelty value and, for Emma's benefit, a demonstration of linkages and rotary & linear motion. Sadly, there is little that can be done with the model apart from watching it walk. Any project scripts that you have written to control mBot can be used to control the 'Beetle' variant but there is no need for any new scripts to be written to control it. You *could*, but I think that controlling it with the IR remote in 'manual control mode' is what is intended. It seemed totally pointless for the instructions to specify modifying the line-follower sensor position on the front of the model since the model only vaguely works in this mode until the 'Beetle' needs to turn, which it can't. The avoid obstacles mode also works but once again is totally limited by this robot model's lack of any ability to turn.

It is truly weird in the way that it walks. The biggest limitation of this six-legged robot (and to some extent the other versions too) is the surface on which it walks. The 'Beetle' has difficulty turning on carpets and has no traction at all on slippery tiled surfaces.

I experimented by adding wider 'feet' using short (20mm) lengths of garden hosepipe pushed on to the ends of the legs at either end but I don't think that I gained very much extra traction. I couldn't add anything to the middle leg on each side since this is the one attached to the drive wheel crank and there is no gap between the leg and the wheel. The bottom of the leg too rises above the rim of the wheel when the crank is at the top and therefore anything on the leg gets pushed off.



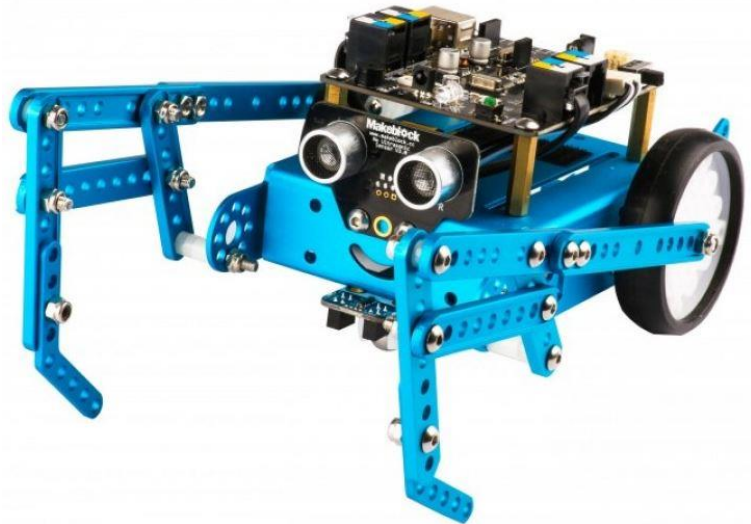


Appendix 8 - mBot Walker Project - Robot 'Mantis'

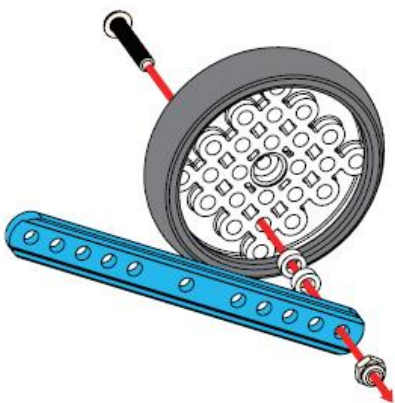
Building / modifying the 'Mantis' model

The 'Mantis' variant of the 'Six-Legged Robot' model is by comparison, a simpler 'walker' than the 'Six-Legged Beetle' and it is also slightly easier to construct.

This model is actually driven by its own wheels and not by its legs. The wheels once again have Screws inserted in them to serve as 'cranks' to operate a linkage on each side (much like the 'Six-Legged Beetle' model). Each 'crank' acts like a cam producing piston like linear motion as the wheel rotates and this then moves the front legs (which look very much as though they are walking and pulling the model along).



Starting with a completely built (fully assembled) mBot, remove the two drive wheels and put them on one side together with the two little self-tapping screws that hold them on to the axles of each electric motor (but leave the tyres on the wheels this time).



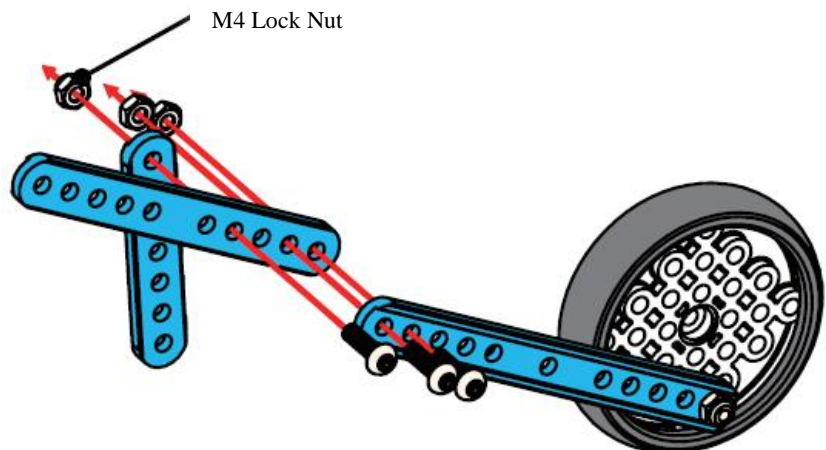
Start by putting an M4 × 22mm Screw through one of the wheels as indicated in the diagram on the left to create the driving 'crank' of this assembly.

Add two 7mm dia. x 3mm thick Plastic Spacers to the Screw on the outside of the wheel and then add a single 11-hole, 108mm 0412 Beam to the assembly using the end hole in the Beam securing it in the usual way with an M4 Lock-Nut, as shown on the left.

Attach a 10-hole, 92mm 0412 Beam to the end of the 11-hole, 108mm 0412 Beam to extend it as shown in the diagram below using two

M4 x 14mm Screws in the two end holes in each Beam and two M4 Nuts to secure them firmly in place.

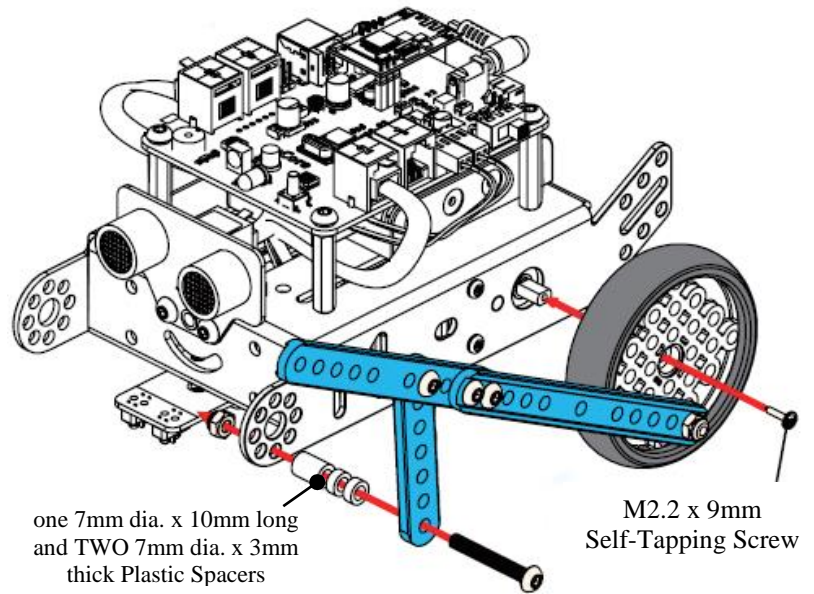
Use another M4 × 14mm Screw (as shown on the right) to attach a 7-hole, 60mm 0412 Beam by an end hole to the fourth hole in the 10-hole, 92mm 0412 Beam with an M4 Lock Nut secured in the usual way for a moving joint - not too loose and not too tight.





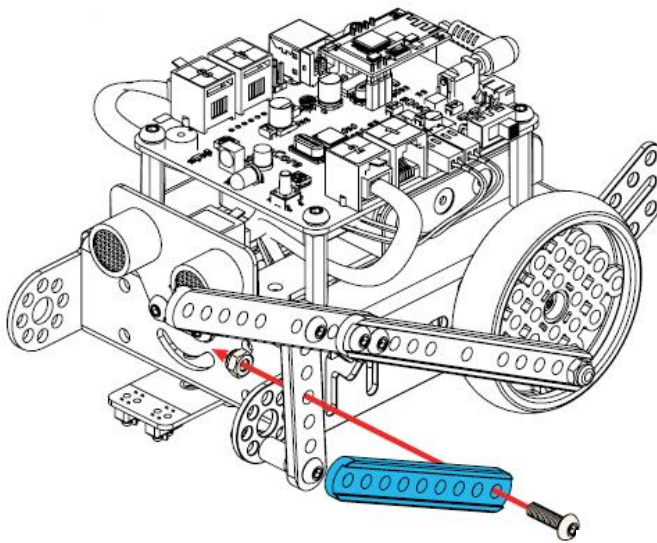
Attach the left-side drive wheel with its new linkage assembly back on to mBot, securing it on to the motor axle using one of the two M2.2 x 9mm Self-Tapping Screws (that you put aside when you removed the wheels earlier).

Using one 7mm dia. x 10mm long Plastic Spacer and two 7mm dia. x 3mm thick Plastic Spacers to (create a 16mm wide spacer unit) use one M4 x 30mm Screw and an M4 Lock Nut to attach the bottom hole of the 7-hole, 60mm 0412 Beam (which you added in the previous assembly step) to the bottom hole at the front of mBots chassis - do note which hole is used for this.



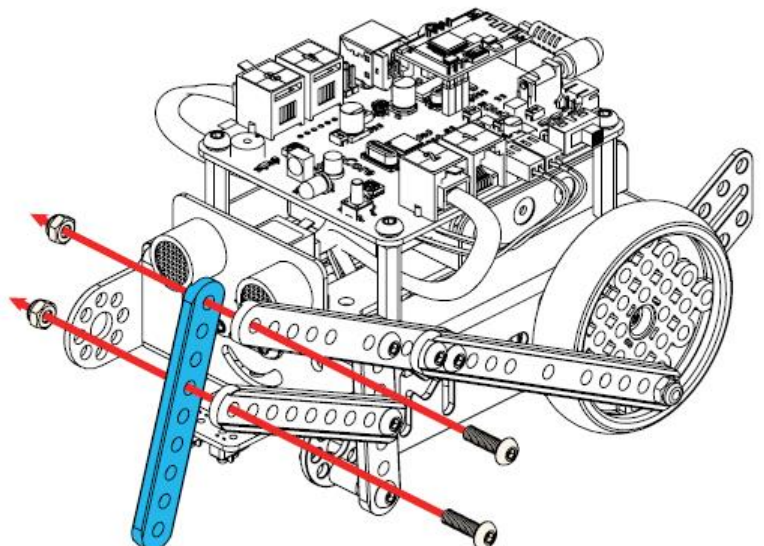
Next, add a 9-hole, 76mm 0412 Beam by its end hole to the centre hole of the upright 7-hole, 60mm 0412 Beam using an M4 x 14mm Screw and an M4 Lock Nut secured in the usual way to create a moving joint. (see the diagram on the left).

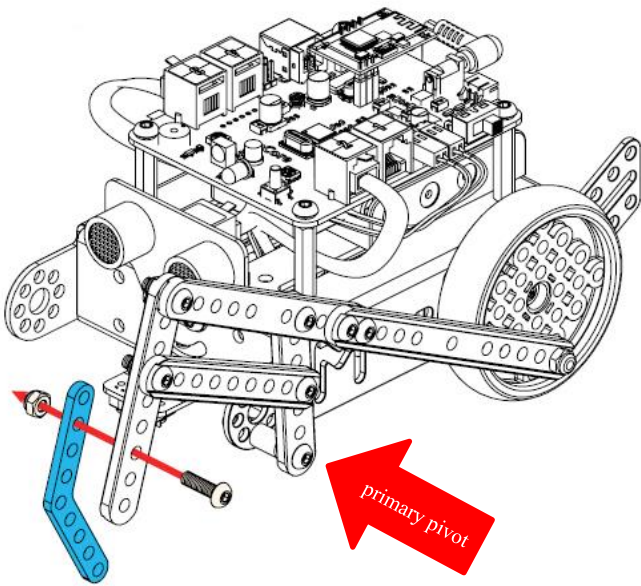
Now, attach a further 9-hole, 76mm 0412 Beam by its end hole to the end hole of the long linkage you made earlier (which you connected to the wheel) using an M4 x 14mm Screw and an M4 Lock Nut secured in the usual way to create another moving joint. (see the diagram below).



Then create another moving joint using another M4 x 14mm Screw and an M4 Lock Nut connecting the fourth hole down in this recently added 9-hole, 76mm 0412 Beam to the end hole in the other 9-hole, 76mm 0412 Beam which you added in the last step above (see the diagram on the right which clearly illustrates this).

The left-side linkage is almost complete. The front leg just needs the addition of a 'claw' (a 9-hole 45° Plate) attached by a single M4 x 14mm Screw and an M4 Lock Nut. Why a moving joint here?

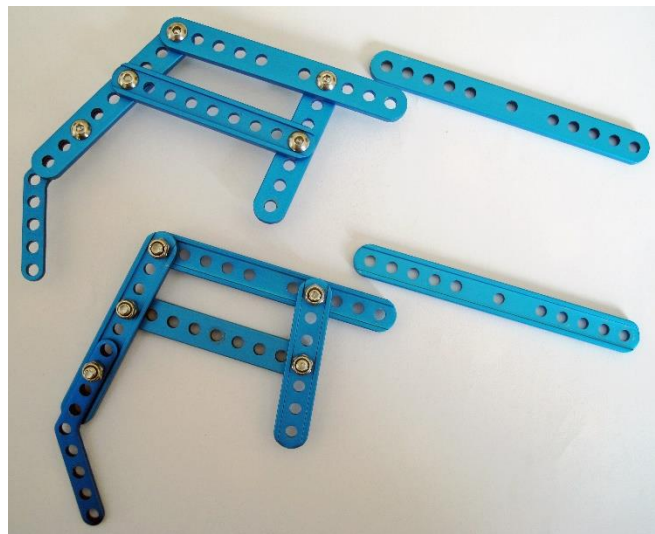




This joint does not move! – but is it meant to? With the ‘claw’ affixed as shown in the diagram on the left, the left-side linkage is complete.

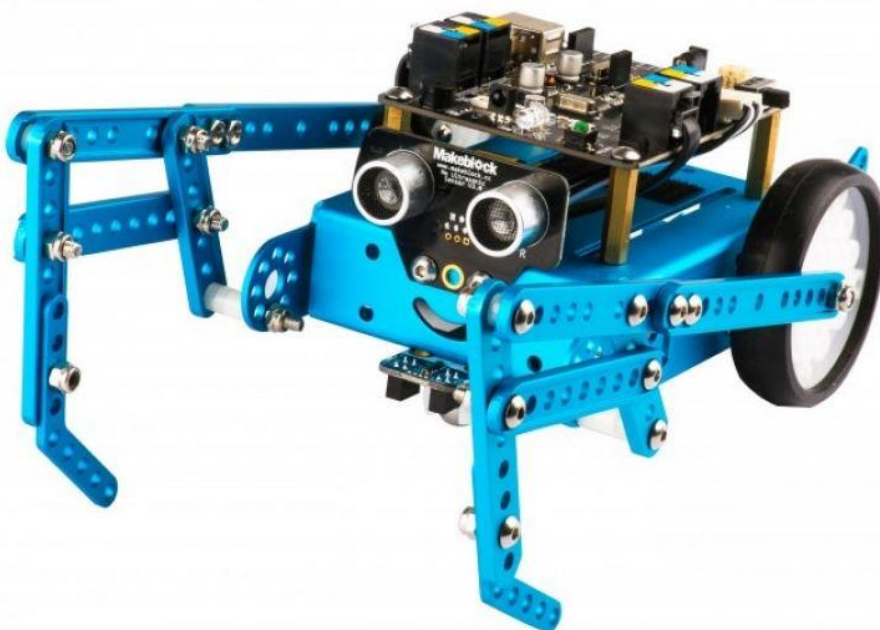
It is not too difficult a task to create a mirror image of this linkage for the right-side of the ‘Mantis’ model. You will assemble it much quicker too.

If I were making this model again, I would build the linkages completely in *reverse* order.



It would be so much easier to create both linkages as flat units (see the picture on the right) and then finally attach each assembled linkage to a wheel before the wheel is put back on to its axle.

The only remaining thing to do after fitting each wheel with the completed linkage would be to create on each side, the primary pivot for each linkage by attaching the bottom hole of the 7-hole, 60mm 0412 Beam (hanging loose from the linkage) to the bottom hole at the front of mBots chassis using one M4 × 30mm Screw with the 16mm thick combination of plastic spacers and an M4 Lock Nut.



The ‘Mantis’ ‘walks’ best if you rotate the wheels manually before starting so that the ‘crank’ bolt on the left-hand wheel is at the front of the wheel (9 o’clock position) whilst the ‘crank’ on the right-hand wheel is at the back (also the 9 o’clock position).

This will ensure that the front claws move forward alternately, and this is much more effective!

Turning the model is not possible, so it’s just forwards or backwards using the IR remote; nevertheless, it’s fun to watch - so just enjoy it.



Appendix 9 - mBot Walker Project - Robot 'Frog'

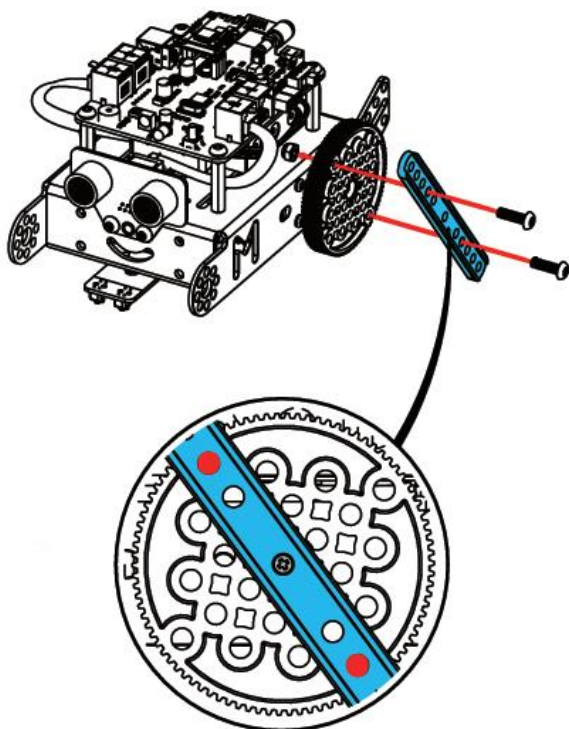
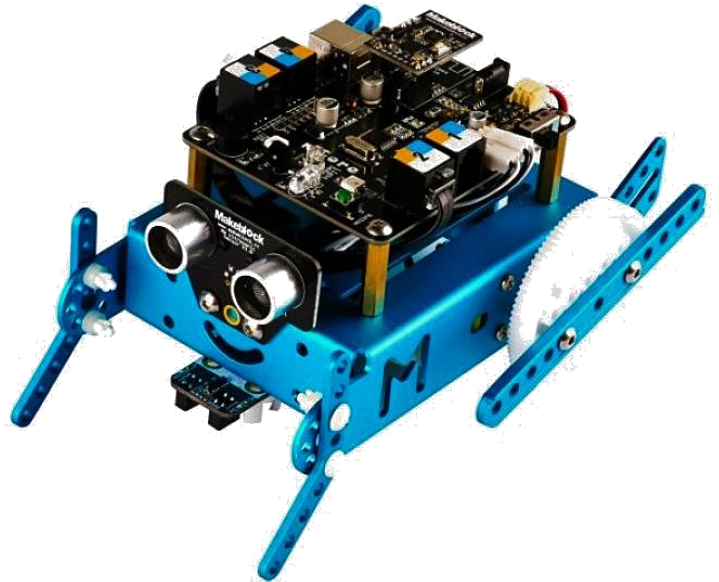
Building / modifying the 'Crazy Frog' model

The 'Crazy Frog' variant of the 'Six-Legged Robot' model. It is a fairly simple construction which is quick to make and possibly the most fun to play with; purely because it is so amusing.

Like the 'Mantis' model, its limited sadly to just forwards & backwards control using the IR remote. Turning isn't a possibility and it works best, incidentally, at full-power.

Emma did think that we could add the sound sensor to the model and write a script to turn both motors on when she shouted "Boo!"

We've done that before, so she knows that this is a possibility - true, but hardly worth the effort when the model goes as well as it does using the forwards and backwards buttons on the IR remote.



Start by removing the wheels from mBot and then remove the tyres from the wheels. Next, attach an 11-hole, 108mm 0412 Beam to each of the wheels as shown in the diagram on the left.

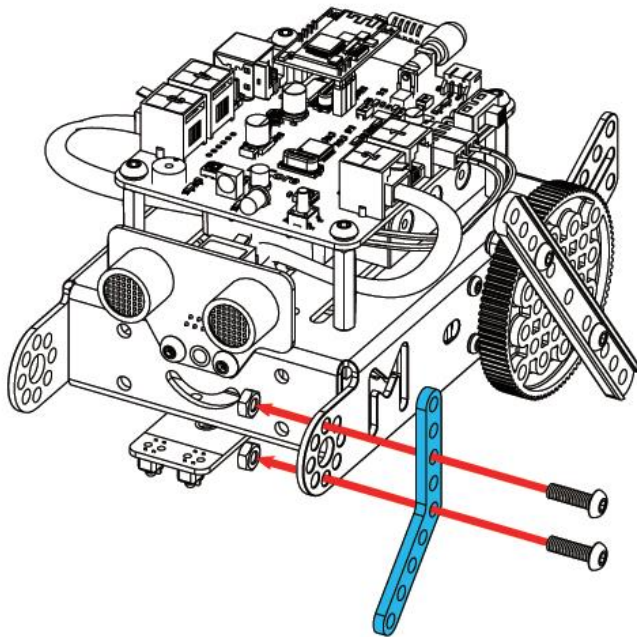
I found that when following Makeblock's instructions to do this that the heads of the Self-Tapping Screws that I put aside when I removed the wheels would not fit through the central hole in the 11-hole, 108mm 0412 Beams.

Even though the Self-Tapping Screws wouldn't, the mBot screwdriver does fit through the holes easily, so I solved this problem by attaching each Beam with just one M4 x 14mm Screw and an M4 Lock Nut to one of the holes in each wheel as indicated in the diagram (but not fully tightened).

Then I pushed each Beam off-centre to expose the hole in the centre of each wheel and insert the M2.2 x 9mm Self-Tapping Screws into each hole in turn and then pushed each Beam back over the central hole with the Self-Tapping Screw safely in place and then secured each Beam in place using the second M4 x 14mm Screw and an M4 Lock Nut through the correct hole in the wheel as indicated in the diagram. Finally, I fitted each wheel back on to the shaped axles which stick out of the motor and through mBots chassis using the two pre-positioned Self-Tapping Screws described above.



mBot and Me a Beginner's Guide



Next use two M4×14mm Screws secured by two M4 Nuts to attach a 9-hole 45° Plate to each side of the front of mBots chassis. The assembly for the left-hand side is shown in the diagram on the left.

When both sides match each other, then the construction is complete and ready for testing using the pre-set 'manual control' routine of button A on the IR remote.

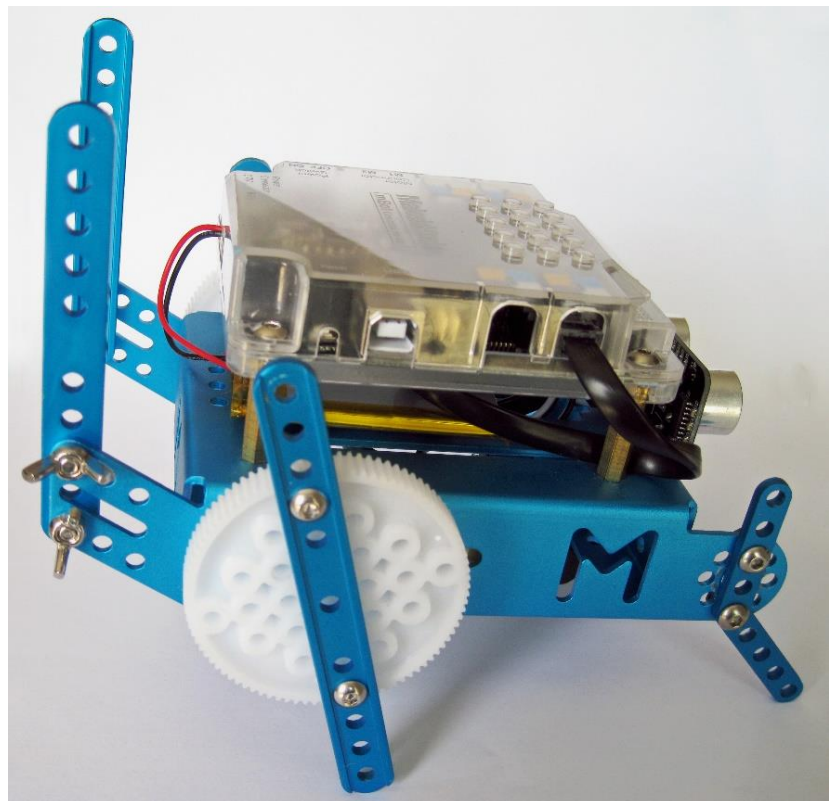
'Crazy Frog' runs with a strange gait if the two Beams attached to the rear wheels are not in alignment but if they both hit the ground at the same time then the 'Frog' can 'hop'. Sometimes it 'hops' with enough force to almost stand upright and very occasionally back-flip and become totally immobile until you pick it up and turn it over again.

It's fun to watch this, but mBot needs modifying with a combination of a motor-cycle sissy bar and wheelie bars like those that extend from the back of a drag-racing car to keep it from flipping over backwards during sudden acceleration.

It's an easy modification to make if you have some spare Makeblock bits to hand.

I used the two 10-hole, 92mm 0412 Beams for this modification (these are part of the 'Six-Legged Robot' model add-on pack and not needed for this model).

I used four M4×14mm Screw to attach them to the rear of mBots chassis as shown in the picture on the right.



You can also see that I used my own M4 wing nuts for speed in assembling / disassembling the modification.

The 'Crazy Frog' model works best at top speed (Button 9 on the IR remote). I suspect that you won't write a control script in mBlock for this - we never did!



Appendix 10 - mBot 'add-on' component - Light/Sound

Makeblock Model No. 98056 - Interactive Light & Sound Pack

The mBot Interactive Light & Sound pack is yet another add-on 3-models-in-1 pack for mBot. The MakeBlock advertising blurb for this says that you can construct either a *'Light Chasing Robot'*, a *'Scorpion Robot'* or an *'Intelligent Voice-Activated Desk Light'*.

The *'Light-Chasing Robot'* model detects the light intensity around mBot using the two light sensors, one mounted on each side of the model. When the light intensity on the left side is greater than that on the right the robot will turn left; when the light intensity on the right side is greater than that on the left side, the robot will turn right; otherwise, the robot will go straight. With minimal changes to the basic model, this modification still allows the full use of the default IR functions built into mBots firmware.

The *'Scorpion Robot'* model gives mBot a scorpion-like curved tail and by adding this to the rear of mBot it alters the centre of gravity, making it easier for mBot to raise its head *and do a "wheelie"* when it powers forwards.

By totally changing its building configuration, mBot can also be turned into an *'Intelligent Desk Light'* with two operating modes. Touch-mode, where the brightness of the light can be regulated by touching the line following sensor with your fingers, and voice-activated mode where the sound level in the area surrounding the desk light is polled and if the volume increases significantly, the light is turned on.

The advertising material also says that each model has infinite possibilities for experiencing with mBot the magic of light and sound. Information in this pack is, however, minimal (see below) - nevertheless if you buy it via China, it only costs approx. £20 inc. free postage - once again an absolute bargain, and mine arrived trouble free in seven days.

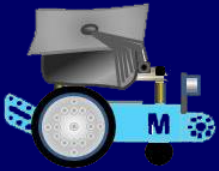
The little manual that comes with the pack (as in the servo pack) shows a sequence of graphics (once again in a similar style to the original mBot manual) showing how to modify mBot to create all three models. It also describes (very briefly) each of the Me Modules contained in the pack.

Thanks to previously searching for the elusive Servo Pack Instructions I knew that I could return to the Makeblock page describing add-on packs (see pages 196 and 197). Here I found the complete model making instructions and some programming examples using the following link:

<http://learn.makeblock.com/en/mbot-add-on-packs/>

As I had found earlier, the programming files were stored as RAR files, so once again I used 7-Zip to decompress them. On examining these mBlock files, I was delighted to see that everything was in English including the comment callouts attached to some script blocks. Written in English, but as usual needing some work to clarify the meaning!

Once again, buying an add-on pack like this will give you a few more extra construction bits to add to your collection. Increasing your collection will enable you to experiment with your own robot constructions (see page 157 to see the collection that I have amassed from add-on packs).



mBot and Me *a Beginner's Guide*

In this add-on pack there are several more useful and wonderful shiny anodised blue (matching my mBot) components. See the illustration on the right:

There are two 5-hole, 72mm 0808 Beams, two 5 x 2-hole, 80mm 0824 Beams and a single seven-hole 45° Plate.

These are enough for building the specified models, but do not add many more bits in total to your own kit of parts.

In terms of other fasteners, you also get in the pack (and there *are* several more of these than actually specified!):

- 8 × M4 x 22 Screws
- 15 × M4 x 14 Screws
- 8 × M4 x 8 Screws
- 9 × M4 Nuts
- 9 × 3mm x 7mm dia. Plastic Spacers

The pack contains another little spanner (the same as the “wrench” supplied in the Servo Pack). There are also two very useful 35cm lengths of Me cable with 6P6C RJ25 plugs at each end - a welcome addition to your kit since the default lengths supplied with mBot are only 20cm long.

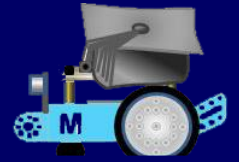
Finally, the included circuitry - The pack contains two Me Light Sensor V1 Modules - these are based on the principle of a semiconductor’s photoelectric effect. They can be used to detect and differentiate between the light intensity in a surrounding area or determine light variances on different coloured surfaces. These sensors can be used to make mBot projects (such as the light chasing robot and the dimming desk light) interact with light.

There is also one very useful Me Sound Sensor V1 Module which can be used to detect the sound intensity in a local area by measuring volume. This is a sensor which gives mBot a “listening” capacity by utilising a microphone and a low-power amplifier combination. This sensor can be used to make interactive mBot projects like a voice operated switch, a voice-activated light or to get mBot dancing in time to musical rhythms.

Finally, there is one Me RGB-LED V1.1 module (exactly the same 4-LED module as supplied in the Servo Pack) so you now have two of these too.

On the whole, another very good value pack of bits and pieces (the above Modules alone would cost in the region of £20 to buy as individual items).





Appendix 11 – mBot & mBlock 5 – All About LEDs

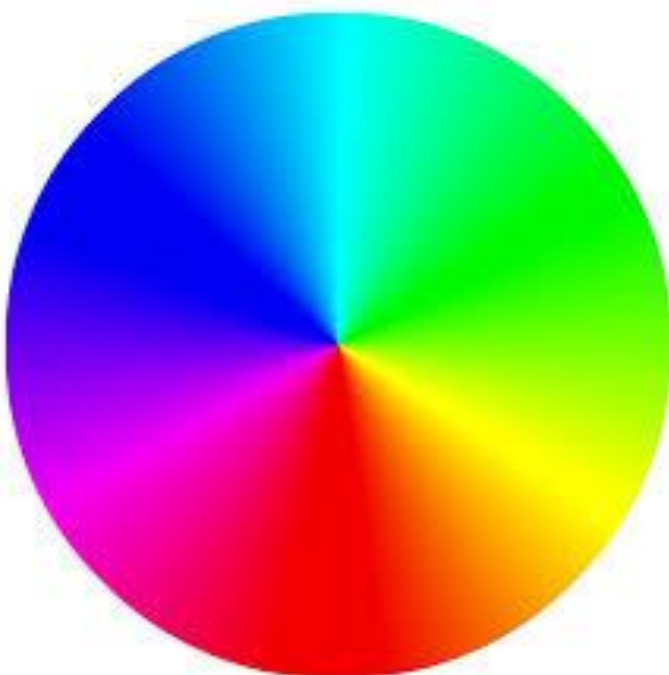
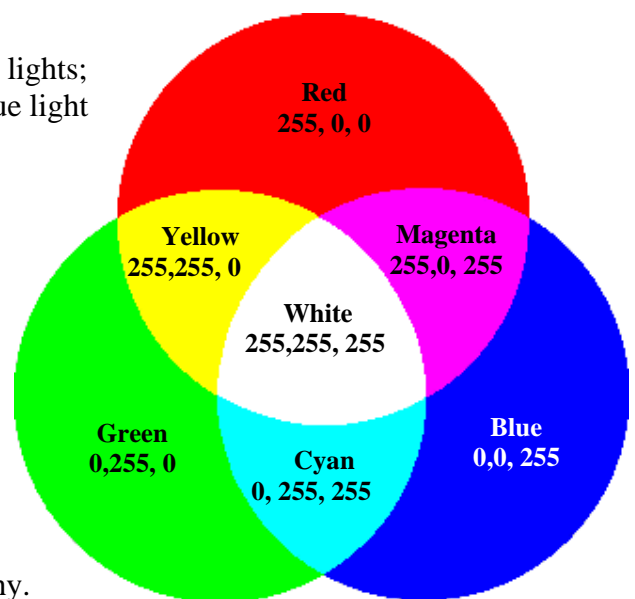
The human eye has receptor cells for three colours - red, green, and blue. Our perception of colour is rather strange in that we see combinations of frequencies as another frequency and that's why these three colours are used in televisions and other visual displays. Multicolour White LEDs are sometimes referred to as RGB LEDs and most of the perceivable colours of light can be formed by mixing them.

A **Light-Emitting Diode** (a **LED**) is a semiconductor light source. RGB LEDs consist of one red, one green and one blue LED and by independently adjusting each of the three they can produce a wide range of colours. LEDs are energy efficient and their disposal causes few environmental concerns. LEDs also have a long lifespan and are subject to very limited wear and tear if operated at low currents and at low temperatures.

White light can be formed by mixing different coloured lights; the most common method is to use **Red**, **Green**, and **Blue** light (see the **RGB** colour model diagram on the right).

Multicolour LEDs can create light of many different colours by mixing different amounts of three primary colours (in a range of 0 to 255) which allows precise and dynamic colour control. The name of the colour model comes from the initials of the three additive primary colours, **Red**, **Green** and **Blue**.

The main purpose of the RGB colour model is for the sensing, representation and display of images in electronic systems such as televisions and computers, though it has also been used in conventional photography.



no copyright infringement is intended with the use of this image

Before the electronic age, the RGB colour model already had a solid theory behind it (based on the human perception of the colours described above). Adding red to green in equal parts makes yellow; adding red to blue in equal parts makes magenta; adding green to blue in equal parts makes cyan; whilst adding equal parts of the three primary colours together yields white light. LEDs have the potential to display around 16 million colour combinations.

The colour-wheel shown on the left sums up the range of colours that can be achieved with RGB LEDs - but remember, you can't really make browns or black at all - you can only make whatever colour you can get when you mix red or blue light, or blue with green light or green with red light!

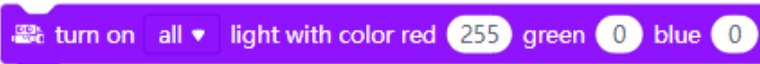


Programming LEDs in mBlock 5

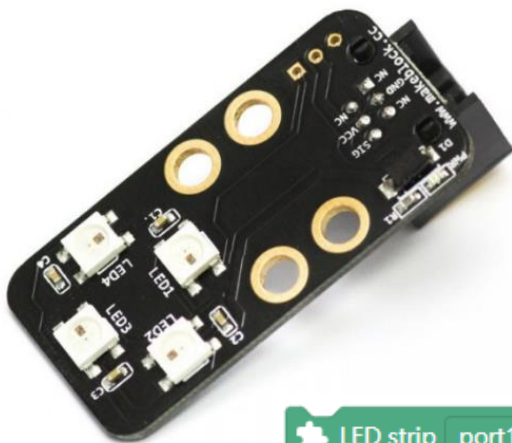
Red, green, and blue colours can be combined (as described on the previous page) by using specific programming blocks in mBlock 5 to produce both white and coloured light from any Makeblock LEDs.

mBots own main mCore board has two 5x5mm full-colour *ws2812 RGB LEDs*. Each of these LEDs has an integrated chip that enables you to control its colour individually by adjusting its brightness value and thereby creating any colour that you want by mixing different amounts of red, green and blue light. Each pixel of the three primary RGB colours can achieve up to 256 levels of brightness and when these are mixed have the potential to display 16,777,216 true colour display combinations!

The second button in mBot's 'Blocks' categories list accesses the 'Show' blocks. Three out of the five 'stack' blocks in this category offer slightly different ways of ways of programming the LED lights on the mCore board. Shown on the left is one of those 'stack' blocks. This block allows you to choose either the left LED or the right LED or both LEDs (*all*) and set the individual RGB values for them.

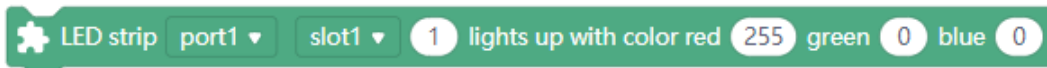


There are two other sets of programming blocks which allow you to control LED lights, but for these you need to load in an 'extension' set of blocks. If you load in the 'Light Sound' extension, then it gives you three 'stack' blocks rather similar to those described above, but this time they are for controlling add-on Me Modules connected to ports on mBot.



Shown on the left is Makeblock's Me RGB-LED v1.0 module, (a component part of both the Servo Add-On pack and the Interactive Light & Sound pack). Makeblock describe this as a cost-effective and easy to control 'shiny' module, ideal for interactive lighting projects.

This module has *four* of the same full-colour RGB LEDs that are on the mCore board and the 'stack' block shown above can be used to set the individual RGB values for each LED individually or all four LEDs at once.



If you load in the 'Makers Platform' extension, then this also has the same light controlling blocks as in 'Light Sound' extension. Additionally though, there are three additional 'stack' blocks and the one shown above can be used via an Me RJ11 adapter connected to mBot (as used to connect servos) to control a Makeblock 1M full-colour flexible silicon LED strip.

These comprise 30 individually programmable RGB LEDs linked into a chain to create 'cool' lighting effects - as shown here on the right:





Each of the 'stack' blocks described on the previous page requires an RGB value (in a range from 0 to 255) to be entered into each of the little windows next to 'red', 'green' or 'blue' in each block.

N.B. mBlock 5 blocks that set RGB values for LEDs do not accept decimals as a brightness value - only integers are acceptable.

The LED lights targeted by your choice of block will mix red, green and blue light in the specified ratio to achieve almost any colour combination. These red, green and blue values are in LED terms the brightness value of each of these colours - **the bigger the value, the brighter and more dazzling the colour!** If all the RGB values in the block are set to zero then the targeted LED will show no colour i.e. it has been turned 'Off'! - Do note too, that **mixed colours seem brighter than any single colour.**

It is also worth noting that when setting LED colour combinations **colours can be more clearly identified if they have lower brightness settings**; and it is sensible (if you want to prevent dazzling) to set each LED to a value *below 40*. This will enable clear identification of both colour and light levels.

However at times, you may want to ignore this advice and just *dazzle!*

To set appropriate colour and light values for Makeblock LEDs you can refer to an RGB colour table (from the internet or perhaps from the colour palettes available in Microsoft applications). I took this a little further and experimented with RGB colour mixes using an Excel spreadsheet.

I *divided* each of the *standard RGB colour table values* (which I had found on the internet) by **6.375** (255/40) and then rounded the result to an integer to determine a value to set the **red, green & blue** values of mBlock's programming blocks with a non-dazzling value (*integers <= 40*).

Another purpose of lowering the values by the division factor described above is to make each output RGB LED change slowly and smoothly.












If you use the **red, green & blue** values from my tables then they should yield clearly discernible and smoothly changing LED colours.

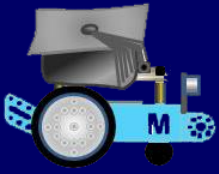
The first of these tables - '**Shades of Red**' is shown here on the right:

The four remaining tables '**Shades of Orange & Yellow**', '**Shades of Green**', '**Shades of Violet**' and '**Shades of Blue**' are shown on the next page.

Each of these tables shows a minimal set of eleven named colours in both RGB LED values and on the right of each table, equivalent RGB graphics component values.

Some Shades of Red:

	Colour Name	R	G	B	R	G	B
	Red	40	0	0	255	0	0
	Tomato	40	10	0	255	69	0
	Deep Pink	40	3	23	255	20	147
	Pale Pink	40	20	26	255	130	171
	Medium Pink	40	16	28	255	105	180
	Pink	40	30	31	255	192	203
	Rose	40	35	35	255	228	225
	Flesh	38	32	27	245	204	176
	Dull Red	32	14	14	205	92	92
	Fire Brick	27	5	5	178	34	34
	Sienna	25	12	7	160	82	45



Some Shades of Orange & Yellow:

	Colour Name	R	G	B	R.5	G	B
	Orange	40	19	0	255	127	0
	Salmon	40	25	19	255	160	122
	Orange	40	25	0	255	165	0
	Gold	40	33	0	255	215	0
	Bisque	40	35	30	255	228	196
	Light Yellow	40	39	32	255	250	205
	Yellow	40	40	0	255	255	0
	Old Gold	34	25	5	218	165	32
	Cool Copper	34	21	3	217	135	25
	Copper	28	18	8	184	115	51
	Bronze	21	18	13	140	120	83

Some Shades of Green:

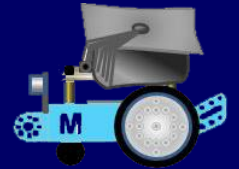
	Colour Name	R	G	B	R	G	B
	Lime	0	40	0	0	255	0
	Lawn Green	19	39	0	124	252	0
	Pale Green	23	39	23	152	251	152
	Turquoise	10	35	32	64	224	208
	Yellow Green	24	32	7	154	205	50
	Khaki	29	28	16	189	183	107
	Sea Green	9	28	17	60	179	113
	Olive	16	22	5	107	142	35
	Green	0	20	0	0	128	0
	Teal	0	20	20	0	128	128
	Dark Green	0	15	0	0	100	0

Some Shades of Violet:

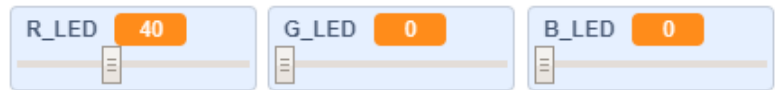
	Colour Name	R	G	B	R	G	B
	Neon Pink	40	17	31	255	110	199
	Magenta	40	0	40	255	0	255
	Violet	37	20	37	238	130	238
	Lavender	36	36	39	230	230	250
	Plum	34	25	34	221	160	221
	Dark Plum	32	7	24	204	50	153
	Orchid	29	13	33	186	85	211
	Purple	25	5	37	160	32	240
	Dark Violet	23	0	33	148	0	211
	Maroon	21	4	15	139	28	98
	Dark Purple	21	4	18	135	31	120

Some Shades of Blue:

	Colour Name	R	G	B	R	G	B
	Blue	0	0	40	0	0	255
	Slate Blue	20	17	40	132	112	255
	Sky Blue	0	29	40	0	191	255
	Light Blue	29	37	40	191	239	255
	Cyan	0	40	40	0	255	255
	Cornflower	15	23	37	100	149	237
	Medium Blue	0	0	32	0	0	205
	Iris Blue	0	28	32	3	180	204
	Steel Blue	10	20	28	70	130	180
	Navy Blue	0	0	20	0	0	128
	Dark Slate	7	12	12	47	79	79



I tested many of these RGB LED value colours in mBlock 5 by generating a new project file in which I created three variables 'R_LED', 'G_LED' and 'B_LED'. I enabled the display of these three variables onto mBlock's 'Stage', giving them slider controls (see the diagram on the right) which allowed them to be quickly adjusted from 0 (min.) to 40 (max.). In then used these values to set the LEDs on mBot to test the 'look' of each colour.



I also created this simple one-block script to activate both of



mBot's on-board LED lights with the values generated by the slider activated variables described above. From these tests I found that my RGB values (all set to be integers below 40 and theoretically reducing the output brightness) would set the colours generated by mBot's LEDs to seem brighter than what might have been expected.

In my colour-mixing Excel spreadsheet file I originally created a table for *greyscale* values, but these LED colours didn't really work in practice, so I deleted that part. It's not really a true statement to say that *LED lights can depict any colour*. The fifty-five shades of Red, Orange, Yellow, Green, Violet and Blue that I have listed all *do* work, but I was rather disappointed with many of the darker shades (especially browns) not really working; even if I proportionally reduced the RGB values even more to make them duller. This is possibly because the closer you get to the upper range of output values; everything becomes whiter and individual colours are harder to discern.

If you set all three RGB colours to full brightness you will get something close to white light, but if you set all three of them to minimum (zero, zero, zero) or even fairly low levels of brightness, you'll get no visibly discernible output from mBot's LEDs!

If the voltage to a LED is lowered, the brightness *will* go down; but lower the voltage enough, it will simply turn off. It is clear therefore that mBot isn't capable of truly dimming a LED, because a LED can't be dimmed effectively. To emulate the dimming of LEDs, Arduino boards (like mCore) use **Pulse Width Modulation (PWM)** and the microcontroller that they use has several built-in PWMs which can be used to adjust the brightness of a LED via programming; switching it on-and-off so quickly that you don't notice the flicker.

To make a LED look like it's set to **10%** (26/255) brightness, an Arduino board (using PWM) will keep it **ON for 10% of the time and OFF for 90% of the time that the LED is activated**. In your own coding using mBlock 5, in reality you don't have to worry about what is happening here; so just ignore PWM and send a range of voltage values (from 0 to 255) that correspond to the brightness levels that you require.

Footnote: *Despite all of the above, personally I am not overwhelmed by the output from RGB LED lights at all. They are OK and they do have their place in projects (and are much better too if seen in the dark) but they are nowhere as exciting or 'cool' as they are made out to be!*



mBot and Me a Beginner's Guide

I created the script shown below-right to test that LEDs *could* be dimmed using a script:

The setting-up of the four variables required for this is fairly obvious from the script image shown on the right.

The routine starts with both of mBot's LEDs both illuminated with bright (dazzling) Red whilst Green and Blue have no values set - (R =255, G=0, B=0).

When the script is run the value of the red component is repeatedly reduced by 1 until its value = 0. All of the RGB values in the block are now set to zero and will be showing no illuminating colour at all - mBot's LEDs are now 'Off'. This sequence takes about 30 seconds (approx. 1/10th second for each step in the dimming loop).

```
when clicked
  Set_Variables
  repeat 255
    if Counter < 255 then
      set Counter to Counter - 1
      set R_LED to R_LED - 1
      turn on all light with color red R_LED green G_LED blue B_LED
```

I also created the following script to set the two LEDs on mBot and the four individual LEDs on an Me RGB-LED module to produce random colours and repeat them forever as quickly as possible in a non-stop display of ever-changing light patterns:

```
when clicked
  forever
    turn on left light with color red pick random 0 to 40 green pick random 0 to 40 blue pick random 0 to 40
    turn on right light with color red pick random 0 to 40 green pick random 0 to 40 blue pick random 0 to 40
    RGB LED port4 lights up 1 with color red pick random 0 to 40 green pick random 0 to 40 blue pick random 0 to 40
    RGB LED port4 lights up 2 with color red pick random 0 to 40 green pick random 0 to 40 blue pick random 0 to 40
    RGB LED port4 lights up 3 with color red pick random 0 to 40 green pick random 0 to 40 blue pick random 0 to 40
    RGB LED port4 lights up 4 with color red pick random 0 to 40 green pick random 0 to 40 blue pick random 0 to 40
```

I needed something very similar to this script (merged with one of my 'Head-Shaking Cat' routines) to programme the 'Light-Emitting Cat' (see 'Appendix 5 on page 215).

N.B. Sadly, all that I had to go on as guidance for that project was the *IMPOSSIBLE* claim in the promotional blurb for the Makeblock Servo add-on pack which described the 'Light-Emitting Cat' as follows:

"A nice cat brightens up your life - The light pierces the darkness - It's brighter in the sun"

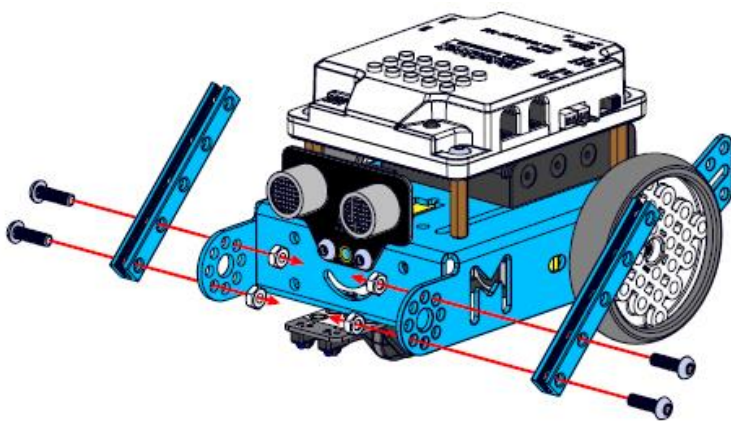
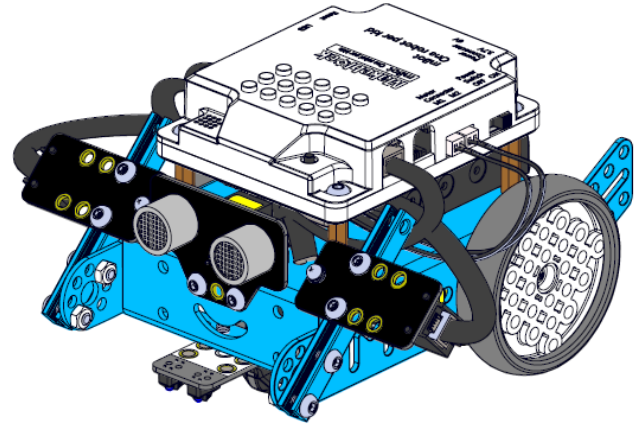
The build instructions for the model show the RGB-LED module mounted on a servo (just like the Me Ultrasonic Sensor module was mounted for the 'Head-Shaking Cat'); but this time mounted on the rear of mBot - like a cat's tail perhaps - or somewhere where the sun does shine!



Appendix 12 - mBot Project - 'Light Chasing Robot'

The 'Light-Chasing Robot' mBot model is a very straightforward and simple modification of the basic mBot configuration. The concept of this model is that mBot can detect the light intensity around it via two Me Light Sensor V1 Modules, one mounted on the front on either side.

Since the feedback from each sensor can be ascertained independently, mBot can determine whether the light intensity on the left-hand side is greater than that on the right-hand side. It can be programmed to turn left towards the light source or, when the light intensity on the right-hand side is greater than that on the left-hand side, it can be programmed to turn right towards the light source. Otherwise, if there is no discernible difference in the sensor readings, then mBot will continue to move forwards in a straight line.

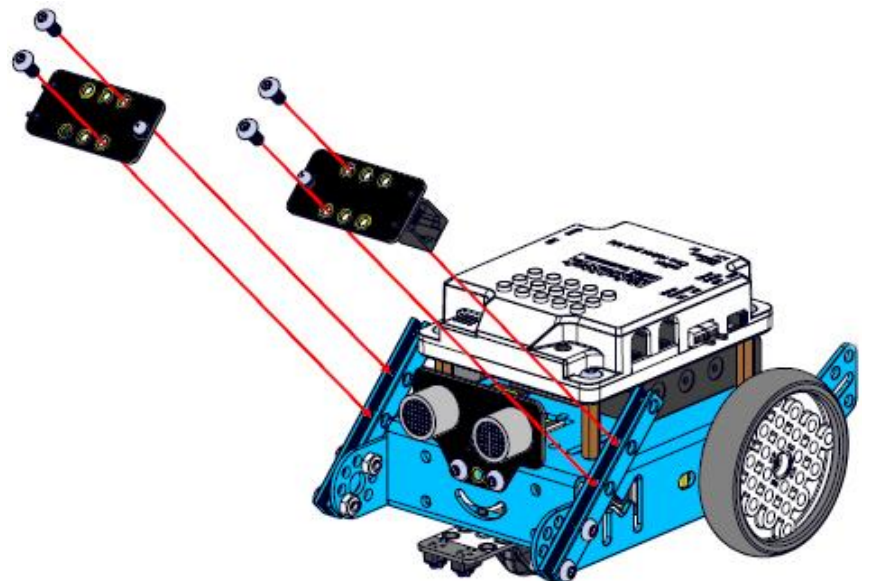


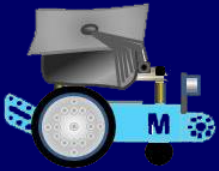
So that's just about it for construction, the fixing of the two sensors. The instructions start by showing the fitting of the two 72mm 0808 Beams that are part of the add-on pack.

These are fitted to the front of mBot with four M4 x 14 screws & nuts as shown in the diagram on the right - do note which holes in the front of mBot are used for this.

Next, the sensors are fitted on to the front of the two Beams with four M4 x 8 screws. These screws can be screwed anywhere in the slot in the edge of the Beams thanks to Makeblock's clever design of the grooved slots.

Try to position the sensors approximately in the positions indicated in the diagram on the right (this is with the screws in line with the third and fourth holes in each of the 72mm 0808 Beams).





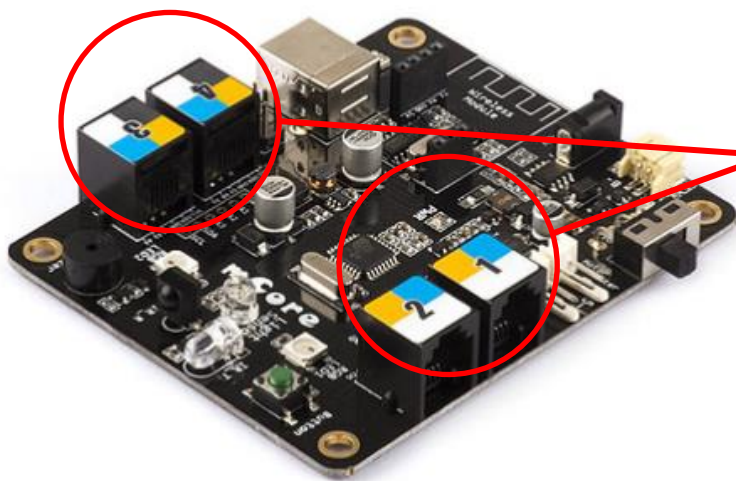
mBot and Me a Beginner's Guide

Not part of this model, but ...

... Since I now had two RGB-LED modules, it gave me the idea of replacing the two light-sensors on the front of the mBot model that I had just built with these two LED modules to create a new model variant of the 'Light Emitting Cat'; but using the configuration of the 'Light-Chasing Robot' model and totally dispensing with the boring servo 'tail' of the original 'cat' model. To do this I simply made a copy of my original 'Light Emitting Cat' project file (see 'Appendix 5 on page 215') and deleted the 'Move_Tail' function and added four more lines to the 'Flash_LEDs' function to flash the LED module on port4 together with the LED module on port3 (as well as the two onboard LEDs). This was a much more satisfying model that showed mBots potential for producing 'disco' light displays.

Now back to this model (and a marginal hiccup in the wiring!) ...

... mBot's mCore main-board has a fool-proof and sturdy RJ25 wiring system providing a simple method of connecting all of Makeblock's Me modules to it. These RJ25 sockets all have coloured labels, but nowhere in mBots little construction booklet does it tell you what these colours mean, or their significance. Once again I had to trawl the internet to find out that the RJ25 connection ports on either side of mBot have different colour-coded combinations.



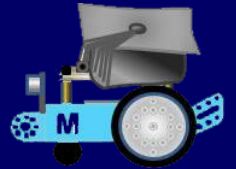
Makeblock's concept is that any compatible Me module must connect to a controller-board port which has a coloured label matching its own coloured label.

Ports 1 & 2 both have three coloured blocks on their labels (the same pattern on each port) whilst ports 3 & 4 on the opposite side of mBot both have a matching checkerboard pattern of four different colours. All of mCore's RJ25 ports have labels showing a combination of yellow, blue and white, but ports 3 & 4 also show a black portion on their labels.

But nowhere is this made very clear! It is very easy to connect one of the *black label* light sensors to the wrong type of port! I got this wrong the first time that I connected this sensor since black labels are on these modules, but confusingly they do not show-up at all well on the black plastic tops of the RJ25 ports! Sound reporter blocks also have a black label and also only report output via ports 3 or 4.

The instructions for the 'Light-Chasing Robot' model clearly show port3 & port4 being used to connect the two sensors to mCore and the mBlock 5 'Reporter Block' for the light-sensor (shown here on the right) only shows port3 & port4 as being available in the drop-down options menu - *and yet I still got it wrong the first time* - because (as is intended) I made the model before programming it!





I should have realised that Makeblock knew best, but in my ignorance, I thought that I could leave the line-following sensor connected to port2 and the ultrasonic-sensor connected to port3 (where they have always been connected since mBots initial set-up). To me, this made considerable sense because that left port1 free on the left of mBot and port4 free on the right of mBot; so connecting the matching light-sensors on each side (left-to-left & right-to right) would be logically very straightforward.

For info. - The controller-board ports are labelled as follows:

- A **Yellow** label indicates a *'One-Way Digital Interface'* which allows the connection of either an Me Ultrasonic-Sensor module, an Me RGB-LED module or an Me-Limit Switch module.
- A **Blue** label indicates a *'Dual Digital Interface'* which allows the connection of either an Me Line Finder module, an Me 7 Segment LED Display module, an Me PIR Motion Sensor module, an Me Camera Shutter module or an Me Infrared Receiver Decoder module.
- A **White** label indicates an *'I²C port'* which allows the connection of either an Me 3-Axis Accelerometer module or an Me Gyro-Sensor module.
- A **Black** label indicates a *'Dual & One-Way Analogue Interface'* which allows the connection of either an Me Light Sensor module, an Me Sound Sensor module, an Me Potentiometer module, an Me Joystick module, or an Me 4 Button module.

and additionally, on other Arduino boards (but not mCore):

- A **Red** label indicates an output voltage of 6-12v DC, which allows the connection of either an Me Motor Driver, an Me Servo Driver or an Me Stepper Driver.
- A **Grey** label indicates a *'Hardware Serial Port'* which allows the connection of either an Me Bluetooth BLE module, or an Me Bluetooth (Dual Mode) module.

On the next page is the marginally modified and transcribed programme which I originally downloaded from Makeblock (as an mBlock 3 file) to operate the *'Light-Chasing Robot'*.

I removed the *'mBot Program'* hat block and replaced it with a *'when (up arrow) key pressed'* hat block intending to run the programme via a Bluetooth connection rather than (as it is in the original) being uploading into Arduino for *off-line* use.

I made two Variables called *'Light_Left'* and *'Light_Right'* to store the feedback from each of the two Me Light Sensor V1 Modules used in the project. I also created a self-defined 'My Block' which I named *'Sensor_Feedback'*. In the main script routine, the single *'Sensor_Feedback'* block is called three times, which by being inside a forever loop gives constant feedback into the two variables shown on the *'Stage'*. You will also note in the programme shown overleaf that I added a simple one-block *'mBot_Stop'* script (using the *'when (space) key pressed'* hat block to halt the robot when necessary.

Remember to connect mBot to mBlock5 using the Bluetooth connection. To operate the model, press the up arrow on the keyboard to start mBot - or hit the spacebar to stop mBot. Use a torch to control mBot's movements by letting it follow (or *'chase'*) the torch-light. When the light-intensity on the left is greater than on the right then mBot will turn left. When the light-intensity on the right is greater than on the left then mBot will turn right. Otherwise mBot will continue to move forwards until stopped.



mBot and Me

a Beginner's Guide

Shown below is the very simple 'Light-Chasing Robot' programme transcribed into mBlock 5 format from the original mBlock 3 file downloaded from Makeblock:

```
when up arrow key pressed
  forever
    if light sensor port3 light intensity > light sensor port4 light intensity then
      left wheel turns at power 70 %, right wheel at power 35 %
      Sensor_Feedback
    if light sensor port4 light intensity > light sensor port3 light intensity then
      left wheel turns at power 35 %, right wheel at power 70 %
      Sensor_Feedback
    if light sensor port3 light intensity = light sensor port4 light intensity then
      left wheel turns at power 70 %, right wheel at power 70 %
      Sensor_Feedback

define Sensor_Feedback
  set Light_Left to light sensor port3 light intensity
  set Light_Right to light sensor port4 light intensity

when space key pressed
  stop moving
  stop all
```

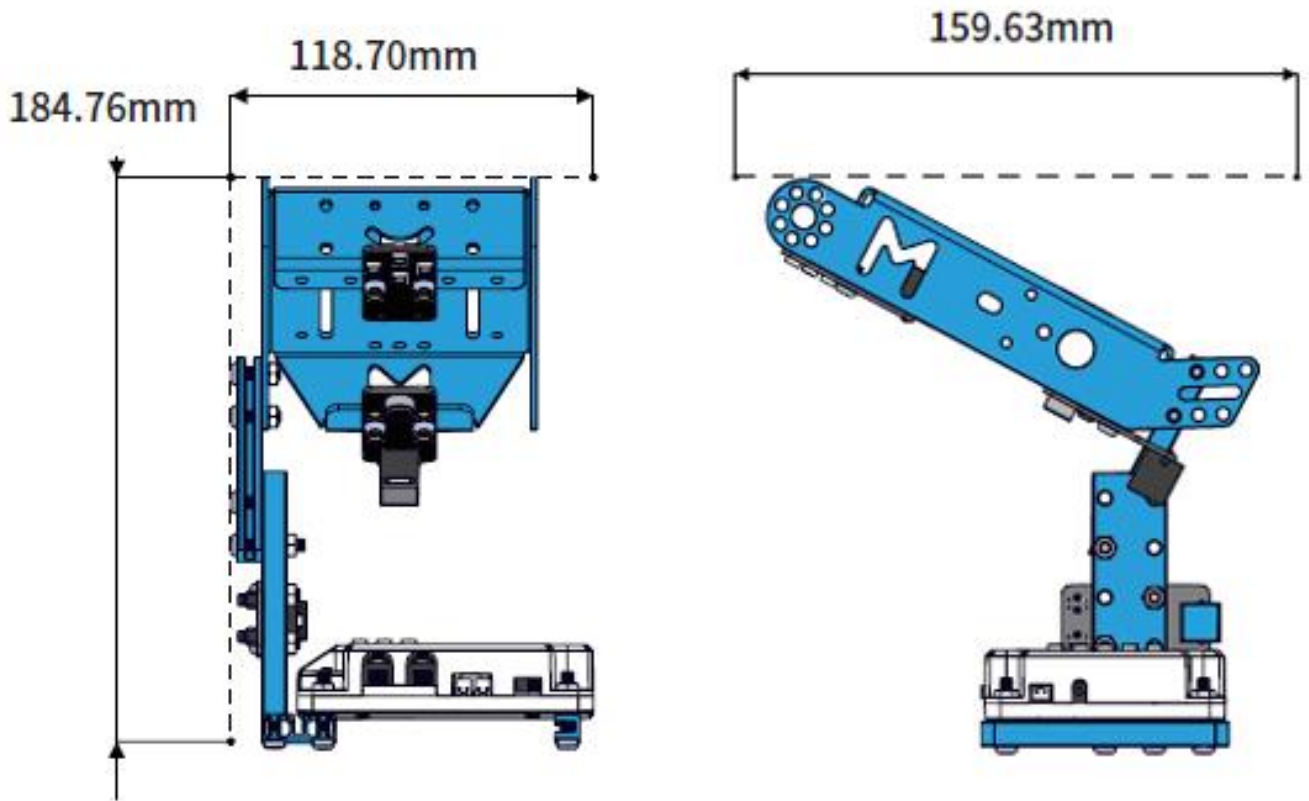
N.B. You could try other power settings here, but 35% and 70% seem to work very well for this model.





Appendix 13 - mBot Project - 'Intelligent Desk Light'

The model shown below (& it's so different to usual!) is what you are going to create next.

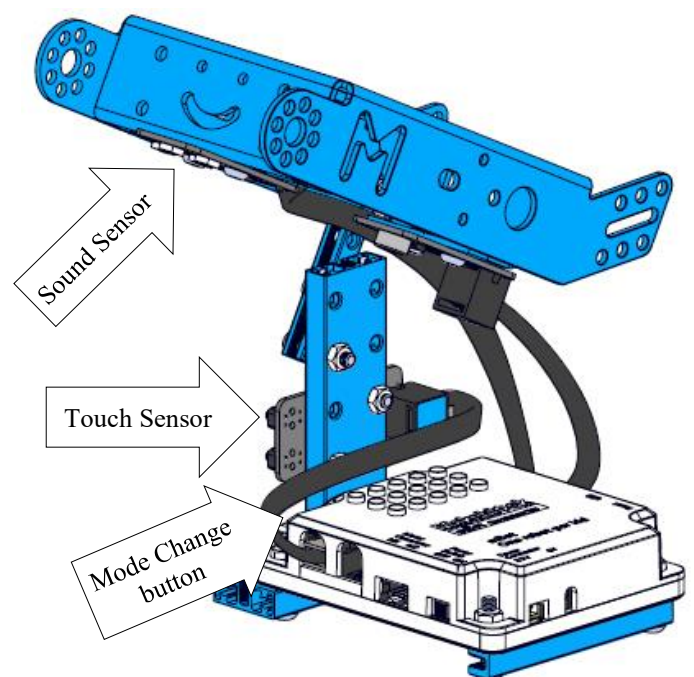


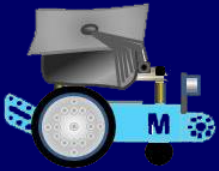
The 'Intelligent Desk Light' mBot modification requires a complete disassembly of mBot and a completely new configuration to be built (see the diagrams above and the illustration below-right).

The resultant desk light has two operating modes; above 'Touch' mode where the brightness of the light can be regulated by touching the line following sensor with your fingers and 'Voice-activated' mode where the sound level in the area surrounding the desk light is polled - if the volume increases significantly then the light turns on.

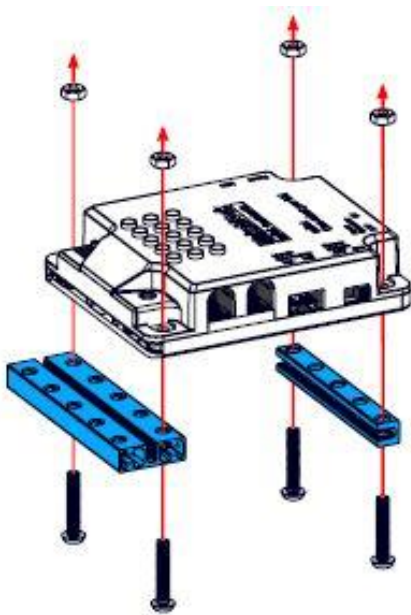
Start by completely disassembling mBot (back to its 'out-of-the-box' state) including removing the drive motors from inside the chassis.

This is a rather disheartening task, but worth it to widen your experiences of what might be achievable by totally reconfiguring mBot components (and perhaps, in time, adding to them any of the other bits that you have gained by buying the add-on packs).





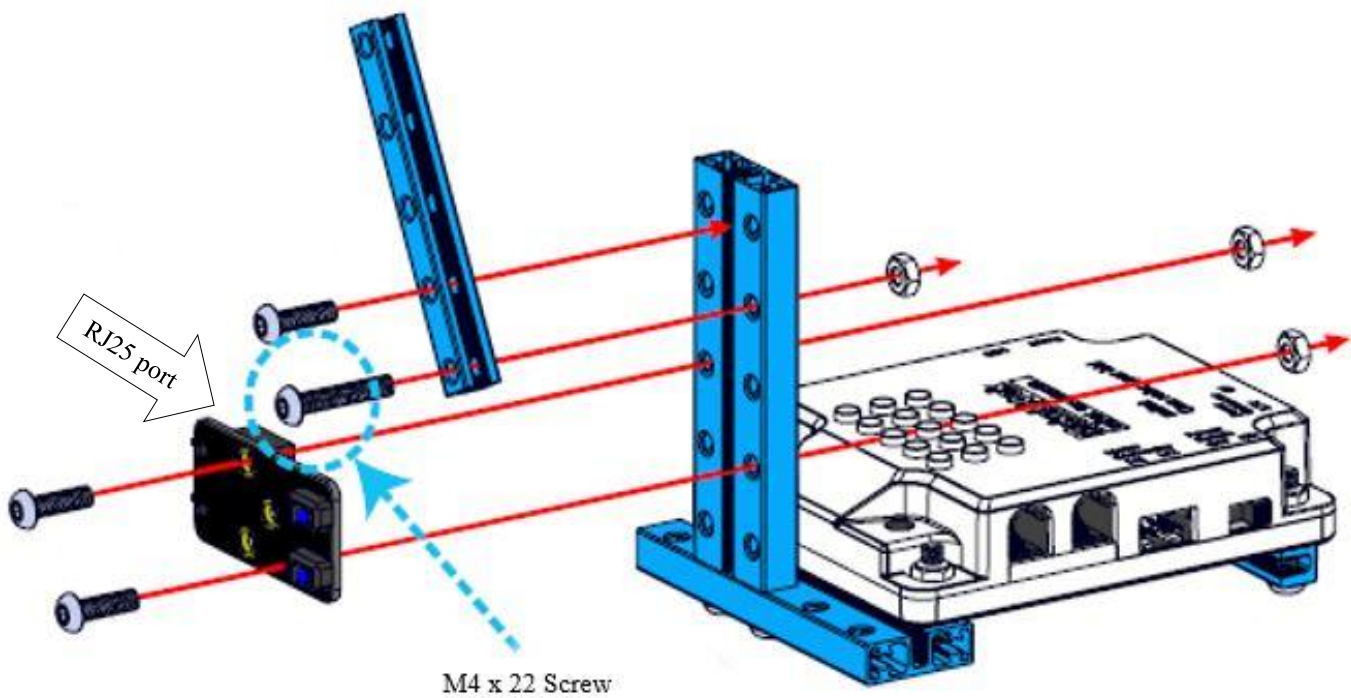
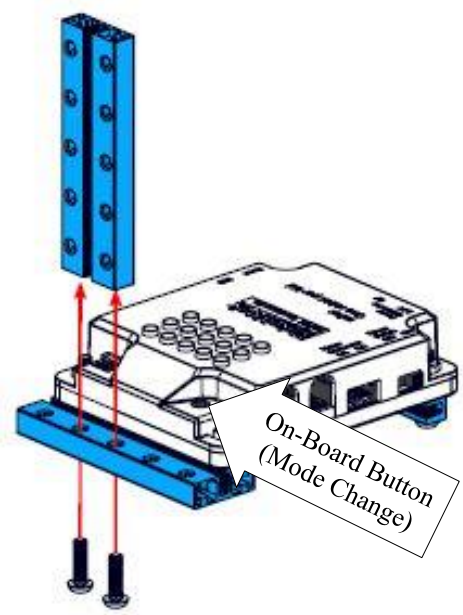
mBot and Me a Beginner's Guide



Using four M4 x 22 Screws and M4 Nuts, attach a 5-hole 72mm 0808 Beam and a 5 x 2-hole 80mm 0824 Beam (as shown in the diagram on the left) to the underside of the mCore board (with its protective plastic case still in situ).

Note that the 0824 Beam is mounted at the front (On-Board Button end) of mCore.

Add a second 5 x 2-hole 80mm 0824 Beam (as shown in the diagram on the right) using two M4 x 14 Screws screwed directly into the end of the vertical Beam.



Next attach to the vertical 0824 Beam the Me Line-Follower Sensor using two M4 x 14 Screws and M4 Nuts (as shown in the diagram above) - do note from the diagram the correct mounting attitude of the module, the holes used and the direction of the RJ25 port.

Also attach to the vertical 0824 Beam, a 5-hole 72mm 0808 Beam as shown at the top of the diagram above. Note that an M4 x 22 Screw and M4 Nut are used through the bottom hole of the 0808 Beam and that an M4 x 14 Screw is used through the next hole which is screwed into the slot the vertical 0824 Beam (so no M4 Nut is necessary here).

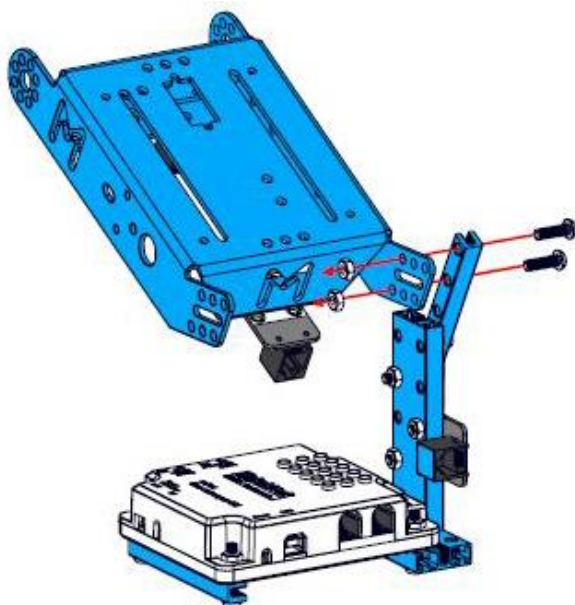
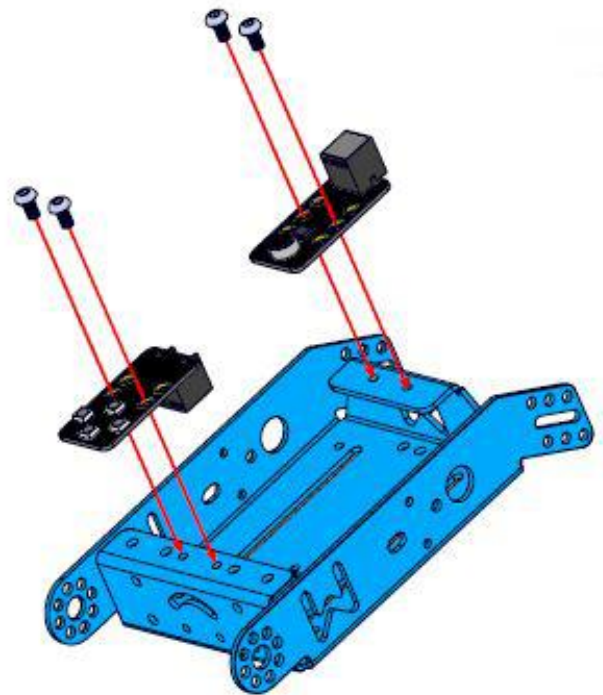


The next stage is to add the Me RGB-LED Module and the Me Sound Sensor Module to the underside of mBots chassis as shown in the diagram on the right.

Note that the Sound Sensor goes at the rear end of the chassis with its RJ25 port pointing outwards whilst the Me RGB-LED Module goes at the front 'mouth' end of the chassis with the RJ25 port pointing inwards.

Use four M4 x 8 screws to do this. M4 Nuts should not be necessary since mBot has threaded holes underneath the 'mouth' at the front and underneath the 'M' at the rear.

Finally, to complete the model mount the chassis unit on to the arm using two M4 x 14 Screws and M4 Nuts (as shown in the diagram on the left).



Wiring: The building instructions suggest the following connections are made: The Me line-follower Sensor is connected to port2, the Me RGB-LED is connected to port3 and the Me Sound Sensor is connected to port4.

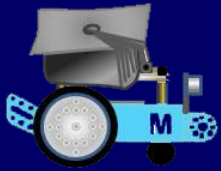
Using the original script Mblock 3 downloaded from Makeblock (and designed to be uploaded to Arduino) you switch the light between its two operating modes by pressing the on-board button on mCore.

In 'Touch Control Mode' (the default setting) the brightness of the light can be regulated by touching one of the Line-Follower's sensors with a finger: Top sensor (light level up) - Bottom sensor (light level down).

In 'Sound Control Mode', the sound in the immediate area is sensed, and if the volume intensity increases significantly then the light will turn on for 5 seconds (at maximum intensity).

However ...

... When I started disassembling mBot, by removing the mCore board and the Line-Follower sensor, I thought it made considerable sense to leave in place the rest of the components fitted to the chassis ready for instant mBot reassembly. The chassis being used as a 'shade' for the desk light is OK, but not really necessary if you use a bit of lateral thinking and improvisation - **a shade is just a simple hollow shape - a box!**



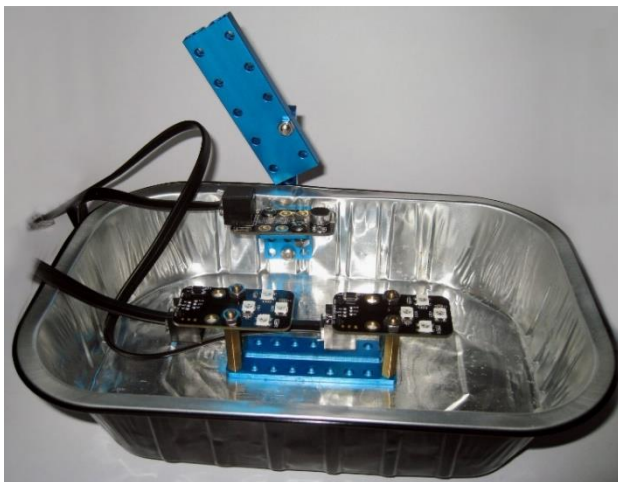
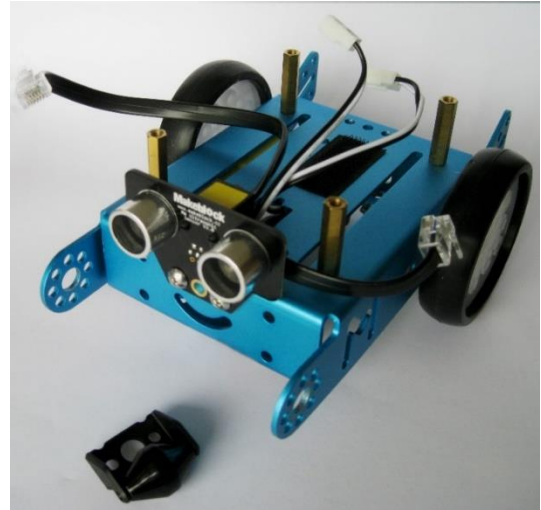
mBot and Me *a Beginner's Guide*

So off I went to our household recycling bin to see what I could find. I had it in mind to use a margarine tub - (not far off in size I thought from the dimensions of mBot's chassis; lightweight & structurally strong too). Sadly, there was none to be found, but I did find a sturdy aluminium foil tray (200 x 120 x 40) that looked fit for purpose. It had contained Moussaka - I think !?!

mBot by this stage was looking forlorn, brainless & immobilised; but it *mostly* remained intact and ready for re-use when required again (see the diagram on the right).

So, after much soaking and careful scrubbing of the tray I had an alternative 'shade' - fairly robust and very lightweight and with the considerable advantages of a neutral black exterior and a shiny reflective interior.

Ideal, if I could reinforce the mounting points (see the picture below) - and, in my opinion, more attractive than the chassis.



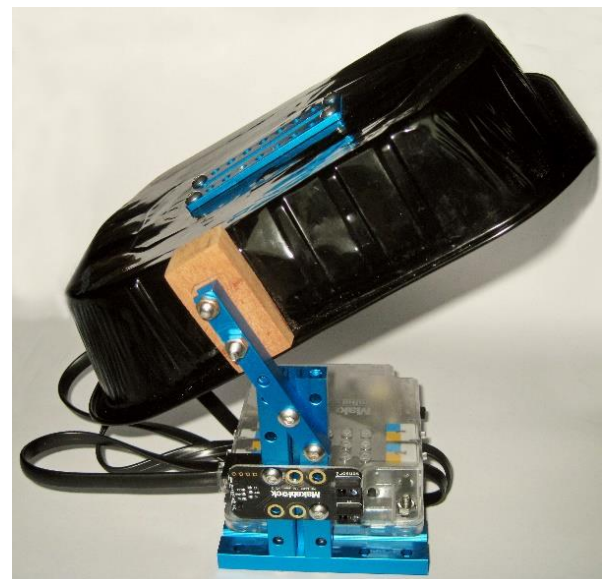
As you can see on the left, I fitted *two* RGB-LEDs end-to-end, each one mounted on a pair of the 25mm tall brass hexagonal pillars normally used to raise mCore up from the chassis (you do get another four of these as part of the Servo add-on pack).

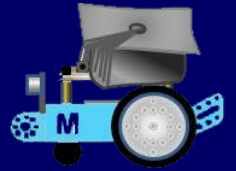
I used both of the 9-hole slotted plates (which also came with the Servo add-on pack) putting one on either side of the foil tray for mounting purposes and using four M4 x 14 Screws to assemble this - it looked very neat, stiffening the tray bottom very effectively.

Next I attached the Me Sound sensor to a 3-hole x 3-hole right-angle bracket (also part of the Servo add-on pack) which I fixed to the side of the tray (see the photographs above-left and below-right):

The idea was to stiffen the wall of the tray on the inside using the angle bracket; and as a stiffener on the outside I made a little wooden spacer (tapered to match the slope of the tray). I used two long M4 screws passing through all of these and through the supporting arm to pull the whole of this assembly tight - it worked very well (see the photograph on the right):

At some point, I still need to trim the wooden spacer down a bit to make it thinner and look neater (and perhaps paint it black); but the whole thing looked quite acceptable and when powered up I was surprised how much reflected light came out from under the foil tray 'shade' too.





Shown below are two views of the complete assembly of my modified variant of the 'Desk Light' model.

Do note from the left-hand picture that if you have a Li-Po battery similar to the one that I specified on page 33 it does tuck away neatly underneath the mCore board - there is also room for the two cables from the two RGB-LED lights to pass below the model too.

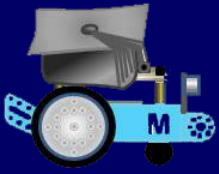


I connected the two Me RGB-LEDs of my model to ports 1 and 2 on mCore (at the front of the model), the Me Line-Follower Sensor was connected to port3 and the Me Sound Sensor was connected to port4 (at the back).

Next I modified the Makeblock script for controlling the desk-light model (see the next page) which I had downloaded by changing all references to 'port3' in all of the 'set LED' stack blocks (there were eight of these) to 'port1'. I then went back to each of these eight stack blocks in turn and duplicated each one, adding the duplicate immediately below the original and setting the duplicate block each time to read 'port2' instead of 'port1'. This was so that the script sequence could control my two Me RGB-LEDs using ports 1 and 2. I also changed the three 'line follower' Reporter block references to 'port3' (leaving the one reference to the sound sensor remaining as being connected to 'port4').

The downloaded script has five Variables: 'Red', 'Green', 'Blue', 'mode' and 'lightIntensity'. I could not see any use for the 'Red', 'Green' & 'Blue' Variables anywhere in the script; so I deleted them (and the script continued to function as it did before). The second modification that I made was rather pedantic, but I wanted to follow the naming convention that I have used throughout the pages of this book.

I renamed 'mode' to 'Mode_Value' and 'lightIntensity' to 'Light_Intensity'. I also modified the monitor readout for the 'Light_Intensity' Variable on mBlock's 'Stage' by giving it a slider to control the brightness of the light if thought necessary.



mBot and Me a Beginner's Guide

The original Scratch 3 script downloaded from Makeblock is very long (see the diagram on the left) and is slightly too long for it to be to display clearly on a single page.

It is also quite hard to follow what is happening here too, so as well as transcribing it into mBlock 5 format I decided to modify it by breaking it down into a short 'set-up' script and several self-defined 'My Blocks' which are called by the set-up script as-and-when they are needed.

The main problem with the original script and my transcribed version of this file (which is shown on the next page) is the use of mBot's on-board button to switch the light activation modes.

When the original script was written in mBlock 3 it had (as you can see here) the 'mBotProgram' hat block at the top, signifying that this programme was to be uploaded

into mBot's flash memory which would have overwritten the 'Factory Firmware' that normally sets the onboard button to control mBot's three default modes of operation.

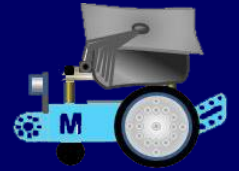
My own transcribed script is instead headed by a 'when (green flag clicked)' hat block because (as usual throughout this book) I have been trying keep all my projects using 'online' programming and avoiding any uploads into flash memory for 'offline' use.

The onboard button will not work in this new script since all it does is start mBots motors and then ignore the rest of the script.

On the next page is the transcribed mBlock 5 version of this project - which can be uploaded into mBot and used 'offline' should you want to do this.

Whilst transcribing, I got to understand how this project works and realising the limitations of the onboard button method made the decision to create a totally new project - something that I had not planned to do in these Appendices at all!

On the next page is my solution to a totally revised and reworked 'Intelligent Desk Light' project ...



```

when clicked
set Mode_Value to 0
set Light_Intensity to 0
set Light_Duration to 2
forever
if when on-board button pressed ? then
if Mode_Value = 0 then
set Mode_Value to 1
RGB LED port1 lights up all with color red 0 green 0 blue 0
else
set Mode_Value to 0
RGB LED port1 lights up all with color red 0 green 0 blue 0
if Mode_Value = 1 then
if sound sensor port4 loudness > 300 then
set Light_Intensity to 255
RGB LED port1 lights up all with color red Light_Intensity green Light_Intensity blue Light_Intensity
wait Light_Duration seconds
set Light_Intensity to 0
RGB LED port1 lights up all with color red Light_Intensity green Light_Intensity blue Light_Intensity
if Mode_Value = 0 then
if Light_Intensity < 0 then
set Light_Intensity to 0
if 255 < Light_Intensity then
set Light_Intensity to 255
RGB LED port1 lights up all with color red Light_Intensity green Light_Intensity blue Light_Intensity
if line follower sensor port3 value = 1 then
wait 0.01 seconds
change Light_Intensity by 5
else
if line follower sensor port3 value = 2 then
wait 0.01 seconds
change Light_Intensity by -5
else
if line follower sensor port3 value = 3 then
RGB LED port1 lights up all with color red 255 green 0 blue 0
wait 0.2 seconds
RGB LED port1 lights up all with color red 0 green 255 blue 0
wait 0.2 seconds
RGB LED port1 lights up all with color red 0 green 0 blue 255
RGB LED port2 lights up all with color red 0 green 0 blue 255
wait 0.2 seconds

```

... but if you still wish to use my mBlock 5 version of the mBlock 3 original script 'offline' (by uploading it into mBot's flash memory) then shown on the left for this purpose is the 'Intelligent Desk Light' script transcribed into mBlock 5 format.

Creating a NEW 'Intelligent Desk Light' Project

I started off by trying to eliminating the 'if (when onboard button is pressed) then' block sequence.

To do this, I removed the forever loop (which was needed to constantly poll the sensors) and experimented by replacing it with much shorter scripts.

I made one script for each lamp mode (*Touch* and *Sound*); each script being triggered by a keyboard keypress and then using a 'repeat until' loop to keep checking for input (just like 'forever') but with the option of another keypress enabling the loop to be exited. A test of this method worked successfully which paved the way for me to start a new project in earnest - **but remember, all of the basic programming for robotics work needs to be done on the 'Devices' tab.**

I created six Variables: 'Light_Adjustment_Factor', 'Light_Duration_Factor', 'Light_Intensity', 'Mode_Text', 'Mode_Value', and 'Sensor_Value' (see more about each of these on the next page).



mBot and Me a Beginner's Guide

The 'Light_Adjustment_Factor' that I created needs to have a value set at project start-up to provide the step value that the 'Touch' light needs to increase or decrease brightness. I intend to set this to a step-unit of 5; but it is very easy to alter this in a 'set (variable) to' block.

The 'Light_Duration_Factor' that I created also needs to have a value set at project start-up to provide the period of time for which the 'Sound' light would remain on. I intended to set this to a very short test value of 2 seconds - but it is very easy to alter this in a 'set (variable) to' block too.

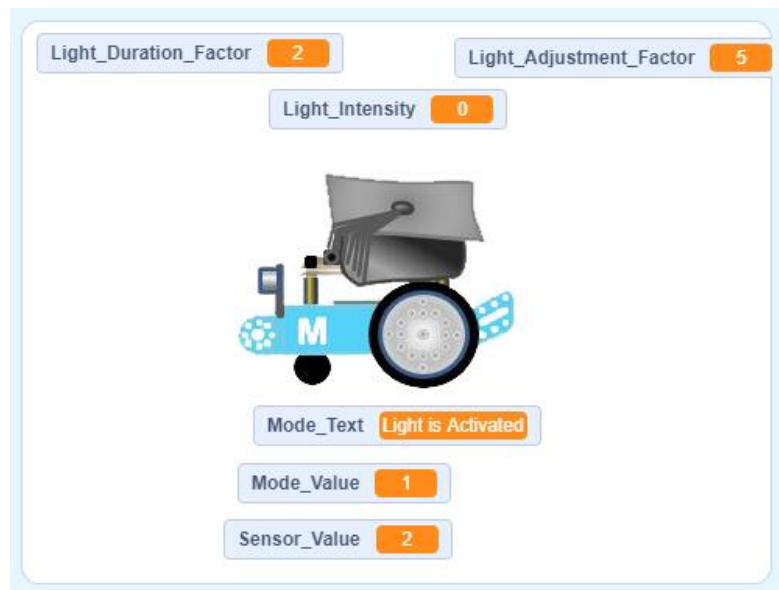
The 'Light_Intensity' variable was renamed from the 'lightIntensity' variable of the original script. This is used to contain a value (0 - 255) which is the brightness of the light. I also set its monitor readout window on the 'Stage' have a slider so that I could manually adjust the brightness of the light.

The 'Mode_Text' variable I added to provide some text feedback to the 'Stage' showing the current mode of operation.

The 'Mode_Value' variable was renamed from the 'mode' variable of the original script. This was designed to contain a value (0 or 1), to act as a switch to change the light operating modes. I intended to use this again (after my a 'repeat until' loop experiments) to switch scripts, but this time using (1 or 2).

I created the Variable 'Sensor_Value' to receive feedback from the 'line follower' Reporter block (connected to port3). This sensor returns a value of 0, 1, 2, or 3 representing four different levels of reflected light and a 'Stage' display of this variable will be extremely useful when testing the sensor.

My initial 'Stage' display of these variables is shown here on the right:



I also defined eight 'My Blocks':
'Controlling_Action_is_Sound',
'Controlling_Action_is_Touch',
'LEDs_Off', 'LEDs_Set',
'Light_Brighter', 'Light_Dimmer',
'Light_Intensity' and
'Light_Soft_Colour_Mix'.

Using these variable names and self-defined blocks I broke my originally transcribed script down into small understandable units:

```
when clicked
set Mode_Text to Light is Activated
set Light_Duration_Factor to 2
set Light_Adjustment_Factor to 5
set Light_Intensity to 0
```

```
when space key pressed
LEDs_Off
stop other scripts in sprite
stop all
```

```
when 0 key pressed
Light_Soft_Colour_Mix
```



```

when left arrow key pressed
set Mode_Value to 1
repeat until Mode_Value = 2
set Mode_Text to Touch Control Mode
set Mode_Value to 1
Controlling_Action_is_Touch
    
```

```

when right arrow key pressed
set Mode_Value to 2
repeat until Mode_Value = 1
set Mode_Text to Sound Control Mode
set Mode_Value to 2
Controlling_Action_is_Sound
    
```

```

define Controlling_Action_is_Touch
Light_Intensity
LEDs_Set
if line follower sensor port3 value = 1 then
Light_Brighter
else
if line follower sensor port3 value = 2 then
Light_Dimmer
    
```

```

define Controlling_Action_is_Sound
if sound sensor port4 loudness > 300 then
set Light_Intensity to 255
LEDs_Set
wait Light_Duration_Factor seconds
set Light_Intensity to 0
LEDs_Off
stop all
    
```

```

define Light_Brighter
set Sensor_Value to line follower sensor port3 value
wait 0.1 seconds
change Light_Intensity by Light_Adjustment_Factor
    
```

```

define Light_Dimmer
set Sensor_Value to line follower sensor port3 value
wait 0.1 seconds
change Light_Intensity by 0 - Light_Adjustment_Factor
    
```

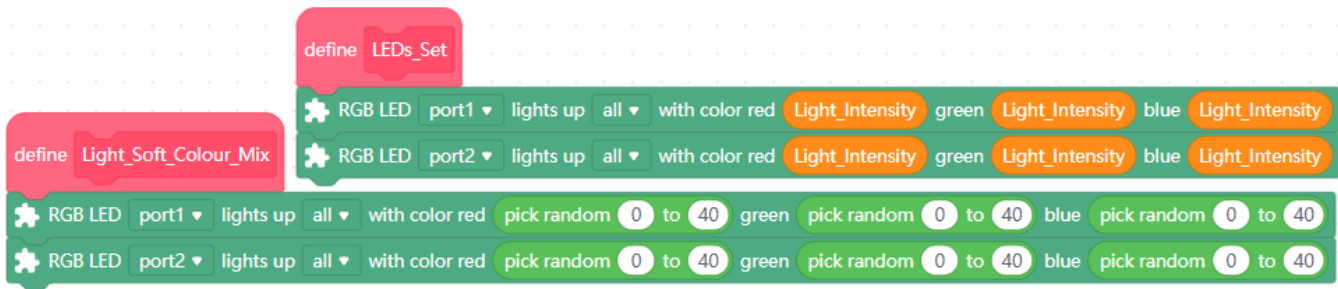
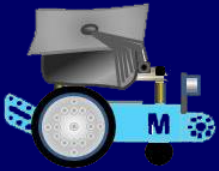
```

define Light_Intensity
if Light_Intensity < 0 then
set Light_Intensity to 0
if 255 < Light_Intensity then
set Light_Intensity to 255
    
```

```

define LEDs_Off
RGB LED port1 lights up all with color red 0 green 0 blue 0
RGB LED port2 lights up all with color red 0 green 0 blue 0
    
```

Shown above (and on the previous page) are most of the scripts that make up my modified and reworked solution to the 'Intelligent Desk Light' project. At the top of the next page are the remaining two scripts which set the LEDs. At this stage of your progress through this book you should find that these scripts are all clearly understandable.



How the modified 'Intelligent Desk Light' project works:

Clicking the 'green flag' activates the project and sets up the values required for several of the Variables. You can change the step value for light intensity changes here as well as the time the light remains on before it automatically turns off in 'Sound' mode.

The left and right cursor keys on the keyboard switch modes and the spacebar turns off the desk-light and stops everything.

Makeblock's interpretation of the 'Intelligent Desk Light' makes a very clever and well thought-out use of the line-following sensor by using its two pairs of infrared LEDs to provide 'touch control' to adjust the desk-light's brightness. See Chapter 12, page 64 for more on how the line-follower sensor works.

N.B. There are a pair of **small blue indicator lights** on the top of the line-follower module and each blue light is a good indicator of what each of the two LED pair components can 'see' and they indicate the following:

- If **both** are **OFF** this indicates low levels of reflected light and the sensor module reports the value '0'.
- If the **bottom** blue light is **OFF**, it shows that low reflected light (a black surface) is detected on the bottom pair of LEDs and the module will report the value '1'.
- If the **top** blue light is **OFF**, it shows that low reflected light is detected on the top pair of LEDs and the module will report the value '2'.
- If **both** blue lights are **ON**, they indicate that there are high levels of reflected light and the sensor module reports the value '3'.

My project solution only uses the returned values of **1** and **2** to determine whether the desk-light's brightness should be increased or dimmed. The original project used the value **3** to make the light flash red, green and blue but since it was easy to activate this by mistake I removed this option altogether and replaced it with the randomly chosen values created in my self-defined 'Light_Soft_Colour_Mix' block which is activated by pressing the zero (0) key on the keyboard.

Should there be a graphical 'front-end' for this project using 'Sprites' tab scripting?

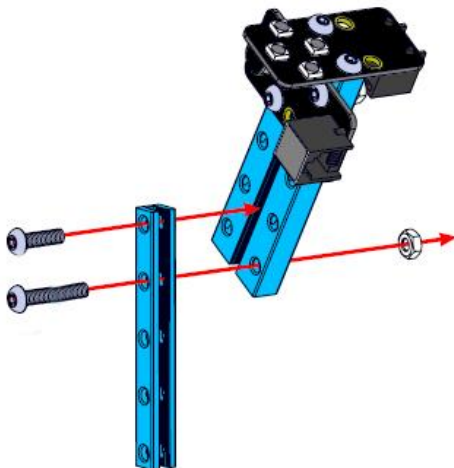
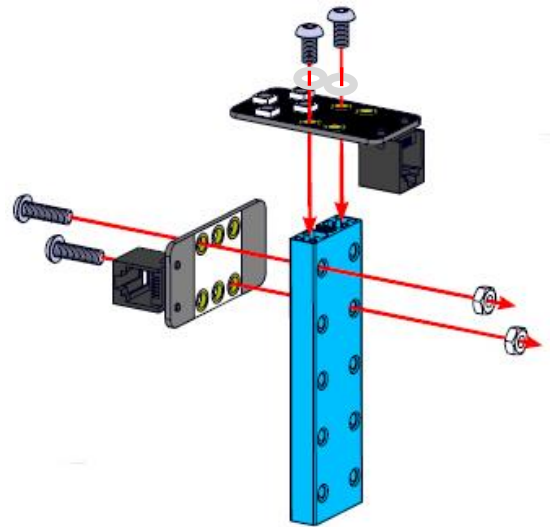
I decided that one was not really needed here - but you can do this using the principles that you have learned earlier if you want to!



Appendix 14 - mBot Light Project - Robot 'Scorpion'

The 'Scorpion Robot' mBot model is another straightforward modification giving mBot a distinctive curved 'tail' that it resembles the tail of a scorpion. By adding the high tail (and two Me modules) to the rear of mBot the centre of gravity is changed, supposedly making it easier for mBot to 'rear-up' and "do a wheelie" when it powers forwards.

The construction of the model works better if you start by attaching the Me Sound Sensor module to the side of a 5 x 2-hole, 80mm 0824 Beam using two M4 x 14mm screws and M4 Nuts (as shown in the diagram on the right). Then add an Me RGB-LED Module to the end of the Beam using two M4 x 8mm screws (I found that I needed to use two 2mm thick Plastic Spacers under the Screw heads, since the 8mm Screws seemed a fraction too long to screw in tightly). Note from the diagram the correct mounting attitude of each module and the direction placement of their RJ25 ports.



Add this assembly to a single 5-hole 72mm 0808 Beam (as shown in the diagram on the left).

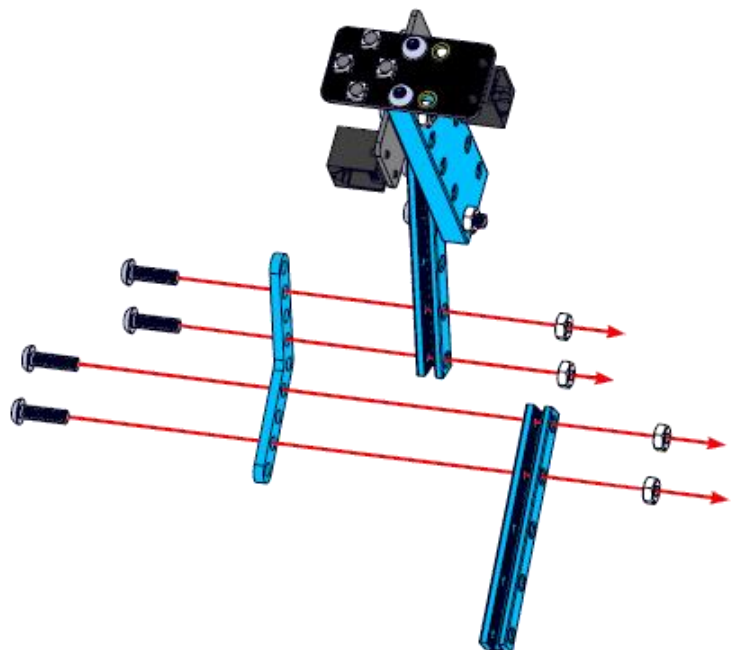
Do note that an M4 x 22mm screw passes through the second hole down the 0808 Beam and is secured with an M4 nut. An M4 x 14mm Screw passes through the top hole in the 0808 Beam and is screwed directly into the slot in the side of the 0824 Beam which has the two Me Modules attached (no M4 Nut is required here).

Next, use the one 9-hole 45° Plate at your disposal to connect your assembly to another 72mm 0808 Beam.

Use four M4 x 14mm Screws and M4 Nuts as shown in the diagram on the right.

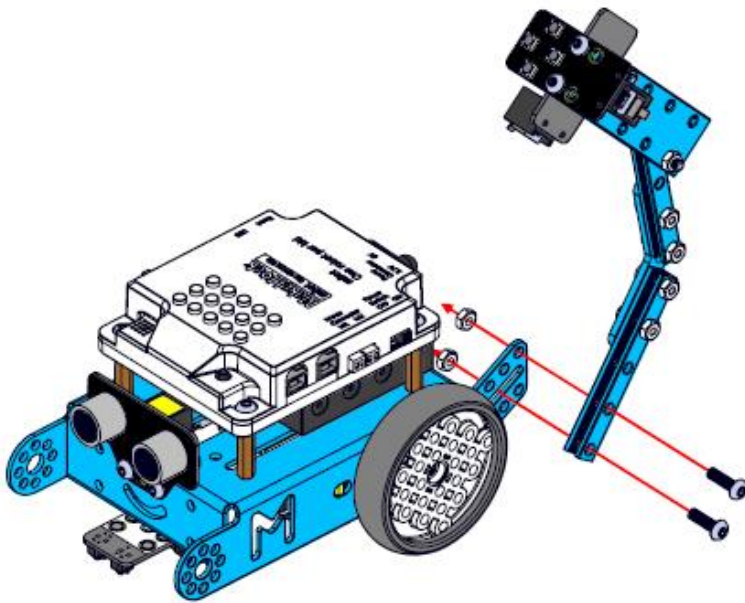
Do note carefully which holes in each arm of the 45° Plate are used for this (the end holes, the centre hole and neither of the holes central to each arm of the plate are not used).

Do ensure that the Me Modules are mounted exactly as shown in the diagram on the right.





mBot and Me a Beginner's Guide



Finally, attach your complete assembly to the rear of mBot's chassis using two M4 x14 Screws and M4 Nuts. Do note from the diagram on the left which holes in both your assembled 'tail' and in the mBot chassis are used for this.

The Me RGB-LED needs to be connected to port1 on mBot and the Me Sound Sensor Module needs to be connected to port4. This leaves the line follower connected to port2 and the ultrasonic sensor connected to port3 as usual. See my notes on Port connections on pages 240 and 241.

On bikes or motorbikes 'wheelies' are the stunt that everyone wants to achieve, but

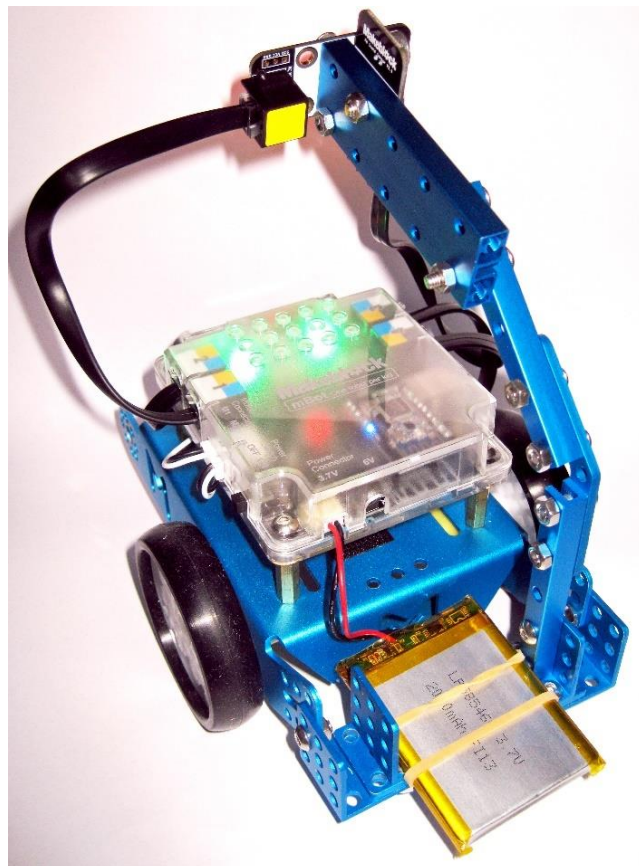
all you're trying to achieve here is a burst of acceleration strong enough to propel the rear wheels forward faster than the rest of mBot, thus lifting the front caster wheel off the ground. To help with this, you need to ensure that the normal centre of gravity of the model is moved further backward. Makeblock's guidelines for this model suggest that you will need to experiment with the position of mBot's Li-Po battery or AAA battery pack (whichever of these you are using) moving them towards the rear and using Velcro to hold them in position.

In reality this minor adjustment made very little difference in getting a 'wheelie' to work successfully so I decided to modify the robot by adding a 'carrier' for the Li-Po battery at the rear of mBot (see the photograph on the right). This was made from bits from two Servo add-on packs. The battery is held in place with a rubber band and positioning it this far back did make quite a difference to the performance.

As you can see I also repositioned the 'tail', connecting it to one of the 3-hole x 3-hole right angle brackets. The 'carrier' is only connected to the chassis by one bolt on either side, so pivoting it slightly fine tunes the balance.

The model works by clapping your hands near the sound sensor, mBot then backs-up a little and waits for another sound signal. It then moves forward briefly and stops and waits for further clap sounds to reactivate it.

Emma particularly liked shouting "Boo" to activate the Sound Sensor Module - in *all* of the projects where it has been used!





```

when clicked
  turn on all light with color red 0 green 0 blue 0
  RGB LED port1 lights up all with color red 0 green 0 blue 0
  set Volume to 1
  forever
    set Sound to sound sensor port4 loudness
    if Sound > 250 then
      RGB LED port1 lights up all with color red 0 green 0 blue 0
      play note C4 for 0.25 beats
      if Volume = 1 then
        set Volume to 0
      else
        set Volume to 1
      if Volume = 0 then
        move backward at power 100 %
        wait 0.2 seconds
        stop moving
        turn on all light with color red 0 green 0 blue 255
        RGB LED port1 lights up all with color red 0 green 255 blue 0
      if Volume = 1 then
        move forward at power 100 %
        wait 0.5 seconds
        stop moving
        turn on all light with color red 0 green 255 blue 0
        RGB LED port1 lights up all with color red 255 green 0 blue 0
  
```

Shown here on the left is the mBlock 3 script downloaded from Makeblock. This has been transcribed into mBlock 5 format and marginally modified.

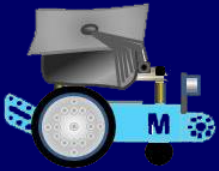
I added a new Variable 'Sound' to display sound level feedback from the sensor on the 'Stage' and this showed (in a quiet room) readings somewhere between 130 & 150.

Thanks to this modification I found that the original sound sensor threshold setting (550) was too high for the script to work but setting it to 250 worked fine.

I altered the 'wait' time (after moving forwards) from to 0.2 secs to 0.5 secs.

Hyped by the advertising blurb, Emma and I both had high hopes and great expectations for this apparently exciting project. Much on a par with the 'Walker' robot models discussed earlier, this very sadly was rather a boring model to both construct and to programme.

Emma thought the project was OK - but I was very disappointed with the overall outcome!



Appendix 15 - mBot components - the latest bits

Several suppliers of Makeblock robotics kits and components in Europe have come and gone in the eighteen months that I have been experimenting with mBot and writing this book. As I have mentioned earlier, many component kits and individual component parts are hard to come by in the UK - unless bought online with quite excessive shipping charges.

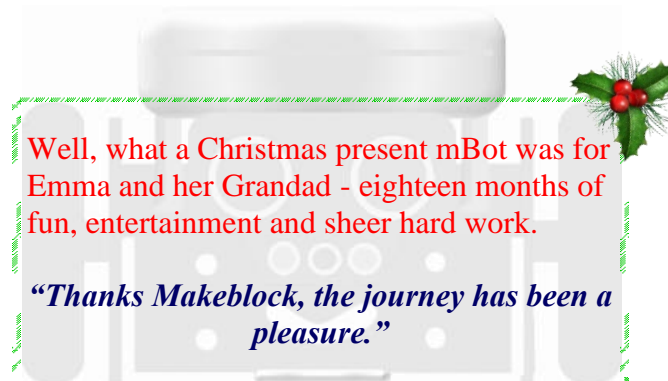
It has therefore been good to see that this year Makeblock have set up a UK storefront in the Amazon-EU Online Store (although all products purchased there still ship from Germany with a fairly hefty £5 shipping charge applied on top of fairly premium prices) - so, costly, but easy to do and safe!

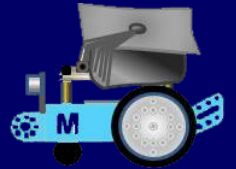
The add-on-packs already addressed in these appendices are however still easily obtainable in the UK at a reasonable cost, but I am unlikely to buy any more even though three new add-on packs for mBot seem to have appeared over the last year.

- I have been almost tempted by the *'Talkative Pet'* pack, but at £43 it's no thank-you! (although it's almost worth it since it contains the hard-to-get Me Audio-Player module).
- The *'Variety Gizmos'* add-on pack at £37 would give me the rather desirable Me 7 Segment Display module and two (v. B) Micro Switches but out of the six models described for construction three are the same *'Cat'* models that can be made with my existing *'Servo'* pack.
- The *'Perception Gizmos'* add-on pack at £46 would give me an 8×16 LED Matrix Display panel and a Sound Sensor, both of which I already have from my earlier purchases; but I would gain a Temperature and Humidity Sensor, a Potentiometer and a little Motor Pack - sadly, one of the five models that is suggested is the *'Intelligent Desk Light'* (from my existing *'Light and Sound'* pack).

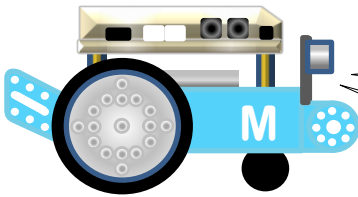
Rather than spending more on these packs, I would much rather spend £125 on Makeblock's *'Electronics Inventor Kit'*. This does have twelve mBot (and mBlock 5) compatible plug-and-play sensors and actuators although it is not an mBot add-on. This kit contains a very useful *'Orion'* main control board that I could programme instead of mBots mCore and this would, I guess, tempt me into moving from mBlock 5 into Arduino programming.

If I chose to spend even more then I would probably upgrade from mBot altogether and go for Makeblock's ten-in-one *'Ultimate Robot v.2.0 Kit'* at £280 - (& Emma gets to keep her mBot!).





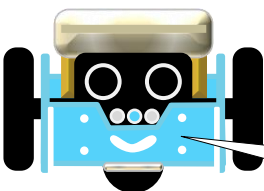
And finally, ...



“Thanks to the powerful software it contains, a computer is a creative tool - use it wisely and *DO* explore the capabilities of your software to the full. A computer is a high-quality graphical interface. *DO NOT* degrade its capabilities, **BE CREATIVE**”.

“Do not be scared of making mistakes and *NEVER* just accept what application software can auto-generate for you - be in control and use it to the best effect to produce **high-quality output**”.

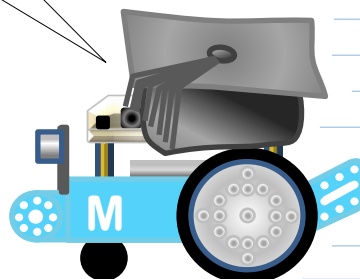
“If it is to communicate to its best advantage; then *everything* that you generate on a computer should be designed to have a **WoW!** Factor”.



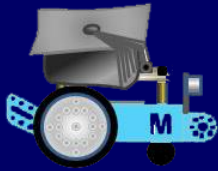
“*So*
... think Laterally
... be Innovative
... Experiment
... Practise
... Plan Ahead
and develop
Good Computer Habits”.

Bye-Bye

But what about programming in Arduino and Python? ...



... **Mañana!**



Index

A

A.I.: 12, 50, 137
Action: 44, 45, 48, 57, 102
Activated: 19, 24, 36, 38, 39, 48, 51, 60, 63, 76, 79, 80, 104, 107, 120, 121, 128, 148, 165, 166, 191, 198, 209, 214, 232, 233, 238, 244, 253
add-on packs: 3, 8, 10, 30, 48, 61, 171, 196, 197, 220, 232, 244, 256
algorithm: 38, 41, 52, 62, 82, 159, 161, 204, 212
alphanumeric: 13, 42, 43, 70, 77, 79, 87, 90, 98, 138, 142
ambient light: 60, 65, 66, 163
analogue: 13, 31, 61, 77, 79, 98, 100, 103, 106, 107, 116, 117, 135, 161
animation: 6, 30, 70, 72, 95, 186, 189
App: 3, 5, 11, 20, 23, 29, 30, 31, 32, 80, 82, 83, 94, 152, 153, 155
Arduino: 1, 4, 5, 7, 10, 11, 12, 14, 18, 19, 25, 33, 35, 43, 102, 151, 198, 238, 242, 246
Artwork: 80, 82
axles: 158, 178, 179, 227, 230

B

Backdrop: 6, 19, 21, 35, 70, 72, 80, 82, 83, 85, 86, 87, 88, 89, 90, 94, 100, 103, 131, 132, 133, 134, 135, 139, 146, 154, 161, 198
Backup: 16, 79, 87, 104, 116, 117, 118, 122, 123, 124, 127, 129, 131, 133, 134, 135, 151, 153
Batteries: 5, 33, 34
Blockly: 30, 32, 52, 80
Blocks: 18, 20, 35, 37, 40, 41, 44, 45, 47, 52, 64, 65, 72, 102, 107, 109, 115, 120, 122, 129, 142, 143, 151, 209, 213, 218, 235, 249, 251
Blocks Area: 15, 28
Bluetooth: 2, 5, 9, 22, 25, 26, 27, 29, 30, 33, 48, 54, 61, 242
Boolean: 36, 40, 42, 46, 47, 49, 59, 63, 149
brass pillars: 172, 177, 179, 180, 199
broadcast: 13, 42, 43, 47, 50, 74, 75, 76, 77, 79, 101, 102, 103, 105, 106, 109, 112, 116, 118, 121, 123, 126, 127, 129, 131, 139, 140, 145, 146, 148, 151, 166, 169, 188
Broadcast Messages: 76, 77, 79, 81, 98
Bubble: 55, 56
Buzzer: 9, 10, 27, 29, 45, 54, 55, 209

C

cable: 1, 8, 10, 27, 156, 174, 193, 194, 248
cam: 50, 149, 222, 227

chassis: 1, 8, 33, 61, 62, 64, 158, 159, 172, 174, 177, 178, 179, 181, 182, 183, 184, 207, 216, 222, 223, 224, 225, 228, 229, 230, 231, 244, 246, 247, 255, 256
Climate Data: 50, 51, 137, 138, 140
Clone Stamp: 84, 94
Cloud: 12, 14, 15, 16, 17, 50, 87, 104, 152, 153
Code: 4, 7, 11, 12, 14, 15, 18, 19, 30, 31, 32, 35, 37, 40, 41, 51, 53, 54, 56, 65, 80, 87, 151, 152, 153
Codey Rocket: 11, 17, 20, 44, 102, 153, 154
COM: 23, 26
Command: 5, 9, 10, 13, 30, 31, 36, 39, 52, 59, 67, 77, 81, 98, 102, 172, 221
Comment: 35, 87, 151, 153, 196, 197, 198, 220, 226, 232
Connect: 19, 20, 22, 23, 24, 25, 26, 27, 69, 107, 111, 124, 201
Connection: 2, 20, 22, 23, 24, 25, 26, 27, 28, 29, 33, 48, 174, 207, 225, 241, 242
Construction: 1, 2, 3, 4, 71, 91, 92, 143, 177, 181, 192, 193, 199, 216, 219, 220, 222, 225, 226, 230, 231, 232, 240, 241, 254
control: 4, 5, 9, 13, 19, 22, 24, 27, 28, 29, 30, 31, 35, 36, 37, 39, 44, 45, 46, 47, 48, 49, 52, 53, 54, 55, 56, 57, 58, 59, 60, 62, 63, 77, 79, 80, 81, 82, 88, 98, 102, 103, 105, 112, 125, 157, 193, 196, 197, 210, 212, 214, 221, 226, 230, 231, 234, 235, 242, 248, 249, 253
costume: 19, 53, 70, 72, 75, 82, 83, 86, 90, 95, 90, 100, 107, 112, 113, 114, 116, 117, 118, 126, 127, 129, 130, 131, 132, 133, 134, 135, 142, 144, 145, 146, 147, 164, 169
Costume Editor: 70, 82, 83, 90, 100, 107, 112, 113, 127, 131, 132, 133, 134, 135
crank: 171, 172, 173, 175, 176, 178, 179, 180, 182, 183, 222, 226, 227, 229
Create: 31
Cursor Keys: 57, 92, 152, 209, 210, 213, 214, 253

D

Data: 14, 43, 47, 50, 60, 61, 86, 101, 105, 116, 144, 145, 164
data-on-demand: 13, 42, 77, 81, 98
default: 3, 7, 9, 13, 14, 15, 16, 18, 20, 21, 23, 24, 28, 29, 30, 40, 44, 47, 48, 52, 59, 61, 62, 70, 84, 85, 86, 88, 89, 101, 102, 104, 105, 106, 128, 144, 148, 149, 150, 152, 153, 154, 155, 232, 233, 246, 249
Delete: 35, 51, 71, 104, 123, 126, 129
Design: 31, 80
Device: 13, 14, 17, 19, 20, 22, 23, 25, 26, 42, 44, 47, 50, 77, 81, 98, 153, 154
Device Library: 14, 20, 153



Device Manager: 23, 26
Devices: 12, 13, 14, 18, 19, 20, 22, 44, 47, 48, 49, 52, 54, 57, 58, 59, 62, 65, 66, 67, 69, 74, 75, 76, 79, 81, 102, 103, 106, 108, 111, 112, 116, 153, 154, 160, 165, 185, 188, 205, 250
Dialogue: 16, 23, 24, 25, 26, 143, 152
dice: 67, 68, 69, 70, 71, 72, 73, 75, 76
different tabs different script: 13, 44, 77, 81, 98
digital: 9, 13, 47, 61, 67, 77, 79, 87, 90, 98, 100, 104, 112, 117, 118, 122, 129, 131
display: 9, 10, 13, 18, 19, 31, 36, 39, 44, 47, 50, 60, 65, 66, 67, 70, 74, 76, 77, 80, 81, 82, 85, 86, 88, 90, 91, 92, 94, 98, 100, 106, 107, 113, 116, 118, 126, 127, 128, 129, 131, 133, 134, 137, 145, 146, 148, 161, 167, 169, 184, 185, 186, 187, 188, 189, 190, 234, 235, 238, 239, 249, 251, 257
Dongle: 26, 27
Driver: 3, 23, 25
duplicate: 53, 114, 126
duplicating: 53, 70, 90, 91

E

Edit: 13, 15, 16, 17, 20, 57, 73, 85, 93, 106, 112, 113, 119, 132
editor: 7, 12, 14, 18, 21, 72, 77, 82, 83, 84, 90, 93, 106, 107, 112, 113, 127, 132, 133, 134, 135, 139, 146, 154
Events: 46, 57, 74, 101, 102, 103, 116, 212
Excel: 6, 68, 71, 95, 96, 236, 238
experiment: 54, 66, 153, 160, 219, 251
Export: 13, 87
Extension: 9, 14, 44, 47, 48, 49, 50, 51, 55, 57, 58, 81, 102, 137, 140, 144, 149, 152, 153, 154, 155, 235
Extensions Centre: 47, 50, 51

F

Factory Firmware: 24
Factory Setting: 9
feedback: 3, 9, 13, 17, 18, 31, 37, 39, 46, 47, 49, 50, 51, 54, 56, 60, 61, 65, 66, 67, 70, 76, 77, 80, 81, 82, 86, 87, 88, 90, 98, 100, 106, 107, 108, 112, 116, 117, 122, 137, 140, 147, 148, 160, 161, 164, 166, 167, 184, 186, 187, 188, 189, 209, 210, 212, 213, 240, 242, 251, 257
files: 13, 15, 16, 17, 20, 21, 22, 43, 48, 52, 55, 71, 82, 83, 84, 85, 87, 89, 93, 94, 95, 131, 151, 152, 153, 154, 155, 190, 196, 197, 198, 220, 232
firmware: 7, 9, 19, 20, 23, 24, 25, 232
flash memory: 7, 10, 22, 25, 37, 54, 59, 81, 217, 241, 249, 250, 253

folder: 16, 87, 94, 152, 153, 154, 155, 197
forever: 37, 39, 59, 60, 63, 65, 66, 128, 129, 187, 191, 210, 213, 239, 242, 250
Forum: 6, 9, 25, 196
Freeform Shape: 71, 73, 91, 92, 164
frequency: 55, 56, 62, 171, 234
front wheel: 158, 159, 222
Full-Screen: 13, 18, 85, 106, 119

G

GitBook: 27, 78
GitHub: 11, 77
Global: 25, 40
gradient: 94, 95, 164
graphic: 9, 13, 19, 20, 21, 37, 42, 44, 70, 72, 73, 75, 76, 77, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 93, 94, 95, 96, 98, 100, 102, 103, 104, 106, 107, 108, 109, 112, 113, 114, 116, 117, 118, 119, 127, 128, 130, 131, 135, 139, 145, 146, 147, 151, 160, 161, 163, 164, 165, 166, 167, 169, 171, 172, 184, 186, 190, 191, 192, 198, 220, 232, 236
Graphics Editor: 84, 90
Green Flag: 18, 35, 46, 53, 54, 59, 63, 66, 76, 103, 128, 139, 146, 148, 151, 172, 191, 249, 253

H

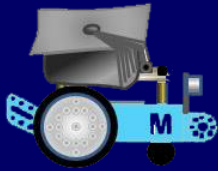
habit: 24, 35, 37, 38, 43, 76, 87, 151, 154
Harmonograph: 171
Hat: 35, 36, 38, 53, 63, 66, 190
high-quality: 6, 8, 13, 77, 79, 87, 89, 95, 98, 151

I

if / then: 40, 59, 62, 63, 114
if / then / else: 40, 59, 62, 63
Infra-Red: 59, 60
Input: 28, 35, 38, 51, 54, 55, 56, 58, 60, 61, 79, 87, 100, 146, 152, 190, 250
interface: 1, 2, 5, 6, 13, 15, 16, 17, 18, 19, 20, 22, 26, 28, 29, 30, 31, 32, 35, 61, 77, 79, 80, 82, 85, 87, 89, 98, 100, 103, 104, 107, 111, 112, 116, 117, 118, 119, 120, 127, 128, 129, 130, 131, 133, 134, 135, 151, 193
IR control: 9, 28, 29
IR remote: 3, 5, 9, 10, 28, 29, 30, 33, 53, 54, 59, 60, 63, 156, 221, 226, 229, 230, 231
Isometric: 71

K

key-presses: 79, 183



L

Learn: 149
LED: 3, 9, 10, 27, 28, 29, 33, 44, 45, 47, 49, 53, 54, 55, 58, 60, 61, 64, 66, 67, 80, 84, 86, 87, 89, 90, 91, 92, 93, 94, 100, 101, 104, 106, 112, 114, 118, 119, 120, 122, 123, 124, 126, 129, 130, 131, 138, 161, 163, 164, 169, 170, 184, 185, 186, 187, 188, 189, 190, 191, 193, 215, 216, 217, 218, 219, 233, 234, 235, 236, 238, 239, 241, 242, 246, 247, 248, 251, 253, 254, 255
Lego: 3, 38, 184, 219
Library: 5, 12, 13, 20, 21, 32, 42, 80, 82, 87, 88, 100, 112, 113, 117, 131, 139, 154, 169
light and shade: 94, 95
light sensor: 10, 60, 65, 66
Light Sound: 47, 49, 235, 45, 49
Light-Sensor: 53
linear motion: 171, 222, 226, 227
Line-Following: 28, 29, 61, 64
Line-Tracker: 53
Linkage: 158, 177, 181, 192, 193, 200, 201, 202, 203, 204, 224
Li-Po battery: 33, 34, 248, 255, 256
list: 13, 14, 15, 20, 38, 40, 41, 42, 43, 44, 45, 47, 49, 55, 57, 58, 62, 68, 69, 74, 101, 102, 106, 109, 112, 113, 114, 115, 116, 118, 126, 131, 134, 136, 140, 149, 153, 185, 186, 190, 221, 235
local: 40, 233
loop: 36, 37, 39, 40, 59, 60, 63, 65, 66, 128, 129, 186, 187, 188, 191, 198, 210, 213, 239, 242, 250, 251

M

M1 (motor): 48, 164, 174, 179
M2 (motor): 48, 125, 164, 174, 179, 193, 194, 206, 224, 228, 230
Machine Learning: 50, 137
Magic Eraser: 84, 94, 164
Makeblock: 1, 2, 4, 5, 6, 7, 8, 10, 11, 12, 14, 17, 23, 24, 25, 26, 27, 29, 30, 31, 33, 35, 44, 48, 50, 55, 61, 70, 77, 80, 81, 85, 86, 125, 137, 152, 153, 157, 158, 171, 174, 184, 192, 193, 196, 198, 207, 209, 212, 215, 217, 220, 221, 230, 231, 232, 235, 236, 239, 240, 241, 242, 243, 246, 248, 249, 253, 255, 257
Maker's Platform: 47, 48, 49
mBlock 3: 11, 12, 13, 15, 21, 22, 23, 24, 25, 26, 46, 48, 50, 57, 63, 81, 86, 125, 151, 152, 153, 160, 161, 188, 197, 207, 208, 209, 212, 217, 242, 243, 249, 250, 257

mBlock 5: 5, 6, 7, 9, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 32, 35, 36, 37, 38, 39, 40, 42, 43, 44, 47, 48, 51, 52, 54, 55, 57, 63, 65, 66, 70, 71, 73, 76, 77, 80, 81, 82, 85, 86, 87, 88, 89, 90, 95, 98, 103, 107, 125, 136, 137, 151, 152, 153, 154, 155, 160, 161, 163, 172, 188, 197, 198, 207, 208, 209, 212, 217, 218, 235, 236, 238, 241, 243, 249, 250, 257
mBot: 1, 11, 20, 22, 66, 146
mBot Ranger: 11, 30, 44, 80
mBot Setup Page: 21, 42, 153, 154, 155
mCore: 7, 9, 10, 25, 27, 28, 33, 45, 48, 56, 60, 172, 174, 177, 179, 184, 193, 199, 201, 209, 210, 216, 217, 218, 219, 235, 238, 241, 242, 245, 246, 247, 248

Me Series Modules: 5, 61

memory: 28, 37, 42, 50, 64

Menu: 13, 15, 85

Message: 9, 13, 42, 67, 75, 76, 77, 79, 101, 102, 103, 105, 106, 108, 109, 112, 116, 118, 121, 122, 123, 124, 126, 129, 139, 151, 164, 166, 169, 186, 188

Microsoft Office: 6, 83

millimetres: 40, 41, 63, 188, 212, 213

Mode: 13, 16, 18, 19, 22, 24, 29, 75, 76, 85, 106, 119, 135, 246

module: 5, 27, 48, 62, 64, 159, 172, 187, 193, 201, 207, 215, 216, 217, 218, 219, 233, 235, 239, 241, 242, 245, 253, 254

monitor: 13, 31, 39, 60, 69, 75, 76, 81, 85, 86, 87, 88, 90, 94, 100, 103, 104, 106, 107, 108, 112, 113, 114, 118, 119, 122, 126, 127, 129, 130, 135, 137, 163, 248, 251

motor: 8, 9, 29, 34, 45, 47, 48, 57, 58, 105, 123, 125, 155, 157, 158, 163, 164, 166, 171, 172, 173, 174, 175, 177, 178, 179, 180, 181, 182, 183, 193, 196, 224, 227, 228, 230, 231, 244, 249

My Blocks: 37, 41, 47, 64, 65, 72, 115, 120, 122, 128, 129, 142, 143, 151, 209, 213, 218, 249, 251

My Projects: 13, 15, 16, 19, 57, 85, 104, 152, 153

N

nest / nesting: 11, 40, 41, 63, 141, 189

O

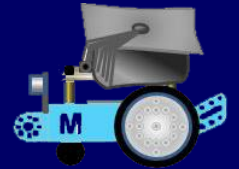
Obstacle Avoidance: 28, 29

on-board: 10, 28, 29, 30, 34, 53, 54, 60, 61, 65, 66, 209, 238, 246, 249

online: 22, 37, 249

Open: 13, 15, 85

Operators: 36, 40, 41, 46, 47, 67, 72, 114, 120, 140, 189



P

panel: 9, 10, 15, 18, 19, 20, 21, 22, 24, 25, 31, 44, 47, 49, 51, 67, 88, 106, 107, 112, 154, 184, 186, 187, 189, 190

Pantograph: 171

parallel: 76, 196

P.C.: 2, 7, 9, 22, 23, 24, 25, 26, 27, 37, 51, 152, 153, 154

Pen: 51, 146, 147, 161, 167, 171, 172, 174, 176, 178, 179, 180, 181, 182, 183

Photoshop: 84, 89, 93, 94, 164

pixel: 88, 89, 185, 235

Plastic Rivets: 182, 221

play note: 55, 56

pointer: 84, 100, 103, 106, 107, 108, 116, 117, 120, 123, 132, 135

port: 2, 23, 24, 25, 26, 27, 33, 34, 125, 164, 201, 207, 241, 242, 245, 246

power: 3, 7, 9, 23, 26, 27, 28, 33, 34, 45, 57, 80, 102, 107, 112, 118, 120, 125, 128, 135, 137, 164, 174, 193, 201, 230, 233, 243

PowerPoint: 6

Presentation Mode: 13, 85, 86

problem: 3, 5, 6, 9, 12, 21, 23, 26, 27, 33, 34, 38, 63, 81, 106, 108, 119, 121, 137, 153, 163, 174, 182, 197, 212, 230, 249

processor: 1, 60

project: 8, 13, 14, 15, 16, 17, 21, 22, 32, 35, 42, 43, 48, 51, 52, 53, 57, 63, 67, 69, 72, 73, 74, 75, 76, 79, 82, 83, 85, 86, 87, 89, 95, 96, 98, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 112, 114, 116, 117, 118, 122, 123, 124, 127, 128, 129, 131, 133, 134, 135, 136, 137, 138, 139, 140, 144, 145, 146, 147, 148, 149, 151, 152, 153, 155, 157, 158, 159, 160, 161, 163, 164, 167, 169, 170, 172, 174, 190, 197, 201, 204, 207, 210, 212, 214, 215, 217, 218, 219, 226, 238, 239, 241, 242, 249, 250, 251, 253, 254, 257

Python: 7, 12, 14, 18, 19, 32, 151

R

Random: 67, 68, 69, 70, 71, 72, 75, 137, 147, 190, 213, 218, 239

realism: 82, 106

real-time: 13, 37, 39, 50, 60, 65, 66, 79, 80, 98, 105, 117, 188

Recognition Window: 50, 149

reflected: 61, 64, 247, 251, 253

Repeat: 39, 40

Repeat Until: 40

Reporter: 18, 36, 39, 40, 41, 46, 47, 58, 62, 68, 69, 119, 123, 128, 131, 137, 140, 147, 189, 241, 248, 251

Reset Default Program: 28, 37

Restart: 19, 20, 23, 25

RGB: 29, 48, 61, 93, 193, 215, 216, 217, 218, 219, 233, 234, 235, 236, 238, 239, 241, 242, 246, 247, 248, 254, 255

RJ25: 8, 61, 193, 194, 195, 199, 201, 206, 207, 216, 217, 219, 233, 241, 245, 246, 254

Robot: 1, 3, 4, 7, 8, 9, 10, 17, 20, 25, 32, 35, 44, 48, 53, 57, 81, 155, 156, 158, 159, 172, 184, 192, 193, 199, 220, 221, 226, 232, 233, 242, 256

Robotics: 3, 4, 5, 6, 9, 10, 11, 12, 14, 17, 19, 20, 23, 29, 35, 44, 46, 47, 50, 57, 67, 76, 77, 79, 80, 81, 82, 88, 152, 153, 155, 157, 159, 160, 185, 250

Rock, Paper, Scissors: 43, 70, 73, 74, 137, 146

roulette: 172, 176, 180

S

Save: 6, 17, 21, 37, 41, 54, 104, 151, 153, 155

say: 67, 70

Scratch: 1, 2, 4, 5, 6, 7, 11, 12, 13, 14, 15, 18, 19, 20, 32, 35, 36, 37, 44, 46, 47, 57, 70, 76, 77, 79, 80, 81, 82, 85, 86, 87, 88, 90, 98, 101, 103, 121, 137, 139, 140, 142, 172, 221, 249

screen-grabbed: 84, 86

Script: 13, 15, 18, 19, 20, 22, 25, 35, 36, 37, 38, 39, 41, 42, 43, 44, 46, 51, 52, 53, 54, 56, 57, 58, 59, 60, 62, 63, 65, 66, 67, 68, 69, 72, 73, 74, 75, 76, 77, 79, 82, 85, 95, 102, 105, 106, 107, 109, 112, 113, 114, 115, 116, 117, 118, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 163, 164, 167, 169, 170, 185, 186, 187, 188, 189, 190, 191, 195, 197, 198, 204, 207, 209, 210, 213, 214, 217, 218, 230, 231, 232, 238, 239, 242, 246, 248, 249, 250, 251, 257

scroll: 9, 55, 109, 116, 186, 187, 189

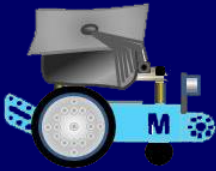
self-defined block: 43, 65, 120, 122, 123, 124, 143, 144, 148, 149, 150, 151, 167, 169, 198, 209, 210, 213, 217, 218, 251

sensing: 44, 45, 46, 51, 63, 107, 108, 128, 137

Sensor: 9, 13, 18, 39, 47, 60, 61, 62, 64, 65, 66, 69, 70, 75, 77, 81, 85, 86, 87, 88, 90, 98, 100, 104, 105, 106, 107, 108, 112, 116, 117, 118, 122, 126, 135, 157, 158, 159, 160, 161, 163, 166, 167, 189, 191, 199, 206, 207, 209, 210, 212, 213, 221, 226, 230, 232, 233, 240, 241, 242, 244, 246, 247, 248, 251, 253, 256, 257

Serial: 3, 22, 23, 25, 26, 27

service bay: 155



mBot and Me

a Beginner's Guide

Servo: 10, 47, 61, 157, 192, 193, 194, 195, 196, 197, 200, 201, 203, 204, 206, 207, 216, 220, 232, 233, 235, 239, 242, 247, 256

set-up: 21, 139, 153, 154, 164

Show: 18, 28, 44, 49, 108, 185, 186, 235

Sign: 17, 22, 76

Software: 4, 5, 6, 7, 9, 11, 12, 13, 14, 15, 17, 22, 23, 25, 26, 27, 30, 35, 50, 80, 81, 82, 86, 153, 188, 197, 198

Sounds: 19, 155

Spirograph: 159, 171, 172

Sprite Library: 80, 82, 89, 90

Sprite: 6, 12, 13, 14, 18, 19, 20, 42, 44, 47, 48, 50, 52, 67, 69, 70, 74, 75, 76, 77, 79, 81, 87, 89, 98, 102, 103, 104, 106, 107, 117, 137, 153, 154, 155, 160, 165, 188, 254

Stack block: 28, 35, 36, 37, 40, 41, 57, 63, 64, 67, 74, 186, 187, 188, 189

Stage: 6, 13, 15, 17, 18, 19, 20, 39, 68, 77, 79, 103, 106, 187, 188, 198, 238, 242, 248, 251, 257

Support Team: 26, 81

T

Tab: 67, 84, 163

Teachable Machine: 50, 137

technical drawing: 71, 91

technique: 28, 41, 79

Text-to-Speech: 51

think: 67, 70

time clock: 128, 129

Timer: 53, 139, 209, 210, 218

Title: 16

Training Model: 50, 149

Translate: 51, 197

transparent: 83, 84, 93, 94, 95, 148, 164, 167, 219

trigonometry: 160

tutorial: 11, 16, 31, 32

U

Ultrasonic Sensor: 8, 10, 39, 40, 60, 61, 62, 100, 105, 106, 112, 117, 157, 159, 160, 161, 163, 170, 172, 184, 188, 189, 199, 200, 201, 206, 207, 209, 212, 213, 215, 217, 239, 255

underscore: 43, 151

understand: 3, 6, 8, 11, 12, 14, 19, 33, 39, 44, 52, 58, 59, 60, 64, 69, 73, 75, 77, 81, 98, 112, 117, 121, 123, 151, 197, 208, 213, 249

Update Firmware: 19, 23, 24

Updates: 7, 9, 14, 17, 24, 25

upload: 7, 13, 21, 22, 25, 37, 50, 54, 81, 82, 83, 87, 89, 95, 112, 131, 198, 242, 250

Upload Mode: 14, 18, 24, 47, 50, 77

USB: 2, 22, 23, 24, 25, 26, 27, 33, 34, 61

user: 6, 11, 12, 13, 15, 31, 35, 37, 39, 40, 41, 77, 79, 85, 154, 186, 190

User Guide: 11

user-defined: 11, 37, 39, 40, 41, 190

V

Variable: 18, 37, 39, 40, 46, 47, 57, 58, 60, 62, 68, 74, 101, 107, 129, 139, 187

variable monitors: 67, 88, 89, 129

variables: 38, 40, 42, 43, 50, 51, 58, 60, 62, 67, 68, 69, 70, 74, 76, 79, 81, 86, 101, 113, 114, 115, 128, 129, 131, 139, 140, 143, 146, 149, 172, 238, 239, 242, 251

VBA (Visual Basic): 6, 95, 96

vector drawing: 84, 93

vectors: 83, 84, 90, 132

version: 2, 5, 6, 7, 11, 15, 16, 17, 25, 27, 47, 55, 57, 69, 73, 85, 87, 95, 145, 146, 152, 160, 171, 172, 176, 177, 188, 190, 191, 199, 207, 210, 212, 249, 250

Video Sensing: 51, 137

W

wait: 60, 66, 75, 160, 163, 186, 188, 210, 218, 257

web-cam: 12, 50, 51, 137, 146, 147, 148, 149

wheels: 1, 53, 155, 157, 158, 171, 172, 173, 175, 179, 181, 182, 222, 223, 224, 227, 228, 229, 230, 231, 255

when key released: 46

when this sprite clicked: 82, 116, 120, 121, 122, 123, 134, 135, 165

While: 40

Wi-F: 5, 61

Wondergraph: 171

Word: 71, 73, 83, 84, 88, 89, 90, 91, 93, 107, 117, 131, 139, 164

Z

Zoom: 94

File Types

.gif: 84, 93

.jpg: 83, 84, 89, 90, 93, 94

.json: 87, 152

.mblock: 14, 152, 153, 207

.png: 13, 50, 71, 83, 84, 89, 93, 94, 95, 100, 112, 131, 138, 164

.rar: 197

.sb2: 152, 153, 197, 207, 217

.sb3: 14, 152

.sprite3: 13, 82, 87, 151, 152, 155, 163, 169



About the Author:

Lindsay Rooms was a teacher at the prestigious Public School, Oundle for 32 years. When he arrived at Oundle in 1976, the school was developing its own computing expertise under the patronage of the Maths and Science departments and Lindsay became interested in the developments in computer programming within the school from 1977 onwards, realising how important it was to industry and engineering. He began to programme in Basic and adapted what he learned into teaching Engineering within the School's workshops system, generating early C.A.D. systems from scratch.

He founded and ran the School's dedicated C.A.D. classroom in 1991 and switched from teaching craft-based skills to teaching Computer Studies / Information Technology in 1994. For the final fifteen years of his teaching career he worked to establish ICT as a subject, teaching GCSE, A Level and the ECDL courses. Lindsay's interest, expertise and widely regarded innovative approach to '*Creative Computing*' resulted in his developing unique techniques to encourage pupils to explore the many possibilities offered by routine computer software.

Deemed an expert in using the Microsoft Office suite of application packages, he was described as, "a teacher who is very good at establishing the basics of good practice; an enthusiast who has developed unusual methods of using application software... a true enthusiast who brought to his department specific expertise in graphic design...".

He retired from Oundle School in 2008, a senior master, where his role as C.O. of the CCF carried its own Head-of-Department status. On his retirement from command in 2003, Her Majesty the Queen appointed Lindsay as a Member of the Most Excellent Order of the British Empire in recognition of his exceptional service as the Contingent Commander of the Combined Cadet Force at Oundle School.

The Naval Secretary's announcement of this appointment in December 2003, stated: "*Since joining the school in 1976 you have made an outstanding contribution to the Combined Cadet Corps, first as Commanding Officer of the Naval Section and, from 1988, as Commanding Officer of the entire Corps. Never flinching from your duties, through hard work and dynamic and inspirational leadership you have had a tremendous and positive impact upon the lives of many young people, gaining for Oundle School CCF a national reputation for the quality of its service. Your tirelessness, enthusiasm and dedication have been remarkable and in the finest traditions of the Service.*"

His role in his final years at Oundle was that of School Proctor. He was a teacher for 38 years but estimates over 40 years of continuous involvement with youth training.

He is married with one grown-up daughter and one grand-daughter and lists his hobbies as Sailing (he is a member of the RNSA), Computing and Gardening. He was until recently churchwarden of his village church.