The MagPi ESSENTIALS

LEARN TO CODE WITH SCRATCH

MAKE SIMPLE GAMES AND APPLICATIONS ON YOUR Raspberry Pi

Written by The MagPi Team

The MagPi Magazine

raspberrypi.org/magpi

THE OFFICIAL RASPBERRY PI MAGAZINE

SAVE UP TO 25%

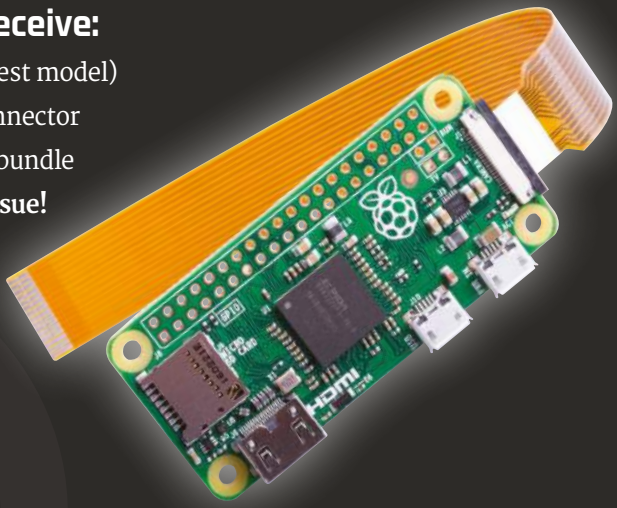
FREE PI ZERO!

Subscribe in print for six or 12 months to receive this stunning free gift

Subscribe today & receive:

- A free Pi Zero v1.3 (the latest model)
- A free Camera Module connector
- A free USB & HDMI cable bundle

Delivered with your first issue!



Pricing

Get six issues:

£30 (UK)
£45 (EU)
$69 (USA)
£50 (Rest of World)

Subscribe for a year:

£55 (UK)
£80 (EU)
$129 (USA)
£90 (Rest of World)

Direct Debit: £12.99 (UK) **(quarterly)**

Other benefits:

- Save up to 25% on the price
- Free delivery to your door
- Exclusive Pi offers & discounts
- Get every issue first (before stores)

How to subscribe:

- **magpi.cc/Subs1** (UK / ROW)
- Call +44(0)1202 586848 (UK/ROW)
- **imsnews.com/magpi** (USA)
- Call 800 428 3003 (USA)

Search 'The MagPi' on your app store:

Available on the App Store

GET IT ON Google Play

WELCOME TO LEARN TO CODE WITH SCRATCH

Coding doesn't just have to be about typing in line after line of gobbledygook. Created by the boffins at MIT, Scratch enables anyone – children and adults alike – to start programming within minutes, without any prior knowledge. You simply drag and drop various code blocks and link them together like jigsaw pieces to form logical scripts, unobstructed by confusing jargon and tricky syntax. Even better, Scratch is included as standard in the Raspbian operating system for the tiny Raspberry Pi computer. It can even be used with the Pi's GPIO pins to interact with electronic components and sensors.

In this book, we'll help you start coding with Scratch, guiding you step by step through the process of creating all sorts of projects: games, animations, quizzes, electronics circuits, and more. It'll be educational and also a lot of fun.

Phil King
Contributing Editor, The MagPi magazine

FIND US ONLINE raspberrypi.org/magpi **GET IN TOUCH** magpi@raspberrypi.org

The MagPi

Available on the App Store

Get it on Google play

EDITORIAL
Managing Editor: **Russell Barnes**
russell@raspberrypi.org
Contributing Editor: **Phil King**
Sub Editors: **Lorna Lynch and Laura Clay**
Contributors: **Sean McManus, William Bell & Code Club**

DISTRIBUTION
Seymour Distribution Ltd
2 East Poultry Ave, London
EC1A 9PT | **+44 (0)207 429 4000**

DESIGN
Critical Media: **criticalmedia.co.uk**
Head of Design: **Dougal Matthews**
Designers: **Lee Allen, Mike Kay**

THE MAGPI SUBSCRIPTIONS
Select Publisher Services Ltd
PO Box 6337, Bournemouth
BH1 9EH | **+44 (0)1202 586 848**
magpi.cc/Subs1

recycle

In print, this product is made using paper sourced from sustainable forests and the printer operates an environmental management system which has been assessed as conforming to ISO 14001.

The MagPi ESSENTIALS

CONTENTS

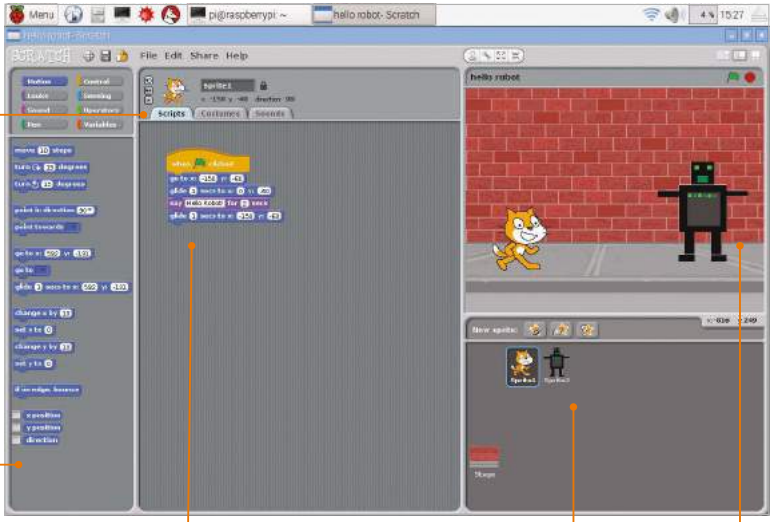
The MagPi ESSENTIALS

[CHAPTER ONE]
GET STARTED WITH SCRATCH

Fancy yourself as Disney or Miyamoto? Whether your inspiration is Mickey Mouse or Mario, Scratch helps you to bring your creations to life…

Tabs: Click the tabs to choose between changing a sprite's scripts, costumes, or sounds

The Blocks Palette: This is where you find the commands to control your sprites. Click the rounded buttons at the top to switch between the different types of blocks



Scripts Area: Assemble your programs here by dragging blocks in from the Blocks Palette and joining them together

The Sprite List: Select your sprites here, so you can change their scripts or costumes. Click the Stage in the Sprite List to add scripts to it or change its background

The Stage: Watch your sprites move and interact here

Get things moving with Scratch! In a matter of minutes, you can build your first program to move the Scratch cat around the screen using the up, down, left, and right cursor keys. When you learn more later, you'll be able to develop this simple program into an art package with the cat as the pen, a game (where should the cat go?), or anything else that needs keyboard-controlled movement. As you work through this chapter, you'll learn how the Scratch screen is carved up, so you can easily find what you need as you build the other projects in this book.

If you're itching to write your own games or start building your own electronics projects, Scratch is the perfect place to start.

Its simplicity comes from the way you select commands from a menu and join them together like jigsaw pieces. Because Scratch comes with a collection of images and sounds, you can start making your first program in minutes.

Scratch's power comes from the many creative ways in which you can combine the commands to make your own program.

[KEEP UP TO DATE]

Get the latest version of Scratch by updating your operating system using: `sudo apt-get update && sudo apt-get upgrade`

[WHICH VERSION?]

If you're using online tutorials, check they're compatible with Scratch 1.4. The newer Scratch 2.0 for PCs and Macs is based on Flash and won't work on the Pi.

Finding your way around

The screen is divided into a number of panes, highlighted in our diagram on the previous pages.

Images that you can control in Scratch are called sprites. You can make them move, draw on the screen, respond to clicks, change their appearance, and interact with each other. A space game might have an alien sprite, a space ship sprite, and a missile sprite, for example. Many projects have more than one sprite, and you can choose between them by clicking them in the Sprite List, in the bottom right. Every new Scratch project includes the Scratch cat.

When you test your program, you'll watch your sprites on the Stage, in the top-right of the screen. Your games are more enjoyable when they fill the screen, though, so when you're ready to play properly, click the easel icon on the right above the Stage to zoom in.

To make your sprites do something, you have to give them instructions that tell them precisely what to do and when. Those instructions come in the form of blocks that join together. The blocks are sorted into eight categories:

Right: Scratch comes with a library of sprites to choose from, including these fantasy sprites

Motion:	Used for moving sprites around the Stage.
Looks:	Used for animating sprites, giving them speech bubbles, and changing their size and appearance.
Sound:	Used for playing recordings or musical notes.
Pen:	Used to draw as a sprite moves around the Stage. Great for making random art, and for special effects in games.
Control:	Used to describe what happens when, and for making bits of your program repeat.
Sensing:	Used to test whether your sprite is touching another sprite or another color, or to get information about other sprites. You can also use the sensor value blocks in your own electronics projects on the Raspberry Pi.
Operators:	Used for maths, random numbers, and doing things to text. There are also blocks here for combining the blocks used in decision making.
Variables:	Used to remember information, such as scores, timer values, or player names.

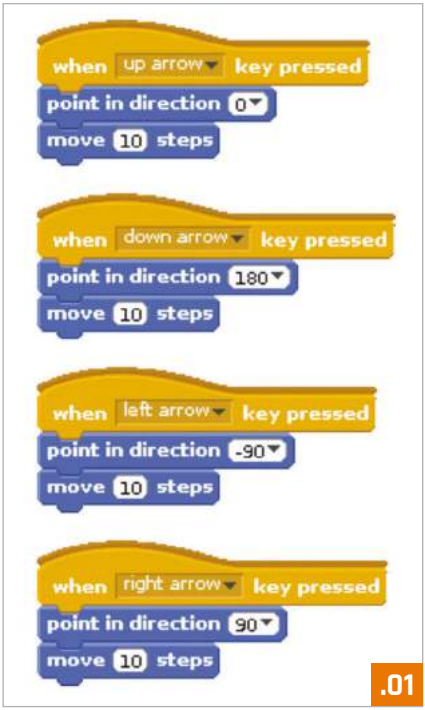
You can find all the blocks in the Blocks Palette on the left of the screen. The blocks are colour coded, so when you're copying programs from books or magazines you can find the blocks you need more easily.

In the middle of the screen is the Scripts Area. This is where you make your lists of instructions (or 'scripts') for your sprites.

Left: The hat blocks in the Control part of the Blocks Palette can be used to start your scripts

[HAT BLOCKS]

The blocks with a curved top, like **when space key pressed**, are called hat blocks. They can only join at the top of a script.



.01

Making your first Scratch script

We promised you could make your first Scratch script in minutes, so here we go!

>STEP-01

Move 10 steps

When you open Scratch (it's listed under Programming in your Start menu), it shows the Motion blocks in the Blocks Palette. Click the **`move 10 steps`** block here and you'll see the cat move on the Stage. Each time you click, it only moves once. That's because '10 steps' is how far it moves, and not how many times. You can click on the 10 and type a different number in here to make it go further or less far with each click. Drag and drop the **`move 10 steps`** block in the Scripts Area.

>STEP-02

Combining blocks

Drag the **`point in direction 90`** block into the Scripts Area. If you drop it just above the **`move 10 steps`** block, they'll lock together. Look for the white line that shows they're about to join before releasing your mouse button. If you click either of the blocks, Scratch will carry out the instructions in order, first pointing in direction 90 (facing right) and then moving 10 steps. Click the Control button above the Blocks Palette. Drag in the **`when space key pressed`** block and join it to the top of your two blocks. Your sprite will move to the right (direction 90) when you press the space bar.

>STEP-03

Making keyboard controls

Right-click your script and choose Duplicate. Click on an empty space in the Scripts Area to drop your copied script. Repeat until you have four identical scripts. Let's turn them into cursor key controls. Click 'space' in the first block to open the menu and choose 'up arrow'. In the **`point in direction`** block below, click '90' and choose '0' (up). Now when you press the up arrow, the cat moves up the screen. Edit the other scripts to add controls for left, right, and down. **Listing 1** shows the finished code.

[GET ARTY!]

Can you add controls for pen up and pen down so you can use this program to draw on the Stage?

The MagPi ESSENTIALS

[CHAPTER TWO]
BOUNCY HEDGEHOG

Spike the hedgehog loves playing on the trampoline, but he's a bit clumsy. Can you move the trampoline to stop him landing with a bump?

In this chapter, you'll make your first Scratch game, in which you use the cursor keys to move the trampoline left and right to catch a bouncing target. This project shows you how to bring in new sprites and backgrounds, and how to use the bracket blocks and diamond blocks in your projects. You'll find these skills useful as you build the other projects in this book. Start a new Scratch project, and get ready to bounce! Remember you can refer back to the last chapter if you need help finding your way around the screen.

>STEP-01

Prepare your artwork

For this Scratch project, you don't need the cat, so right-click it in the Sprite List and then choose Delete. To add a new sprite, click the icon above the Sprite List that shows a folder and a star. Add the trampoline sprite from the Things folder, then the fantasy11 sprite in the Fantasy folder. Let's change the background: click the Stage in the Sprite List and the Costumes tab changes to a Backgrounds tab. Click the tab and use the Import button to bring in your choice of background. We're using the image atom-playground in the Outdoors folder.

>STEP-02

Adding player controls

Click the trampoline (which should be Sprite1) in the Sprite List to select it, and then click the Scripts tab above the Blocks Palette. **Listing 1** shows the scripts you need to add to this sprite. Work your way down them, dragging the blocks into the Scripts Area one at a time and joining them up. Click the white holes in the blocks and type the right numbers in. Remember that the colours are a clue: to find the yellow blocks, click the yellow Control button above the Blocks Palette first.

>STEP-03

Set up the hedgehog

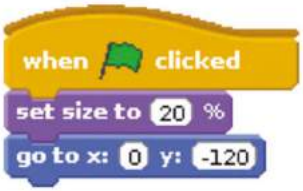
Click Sprite2 in the Sprite List (the hedgehog). Add the script shown in **Listing 2** to it. This puts the sprite in the top left when the game begins, and gives the player a chance to spot it before it moves.

>STEP-04

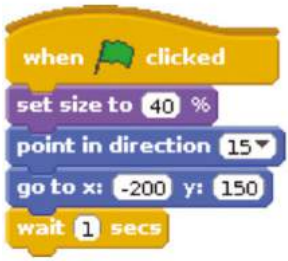
Add a repeat loop

We're going to extend that script now by adding some more blocks at the bottom. **Listing 3** (overleaf) shows the entire script, including the bits you've already done. Click the Control button above the Blocks Palette. Drag a **`repeat until`** block into the Scripts Area and join it to your script so far.

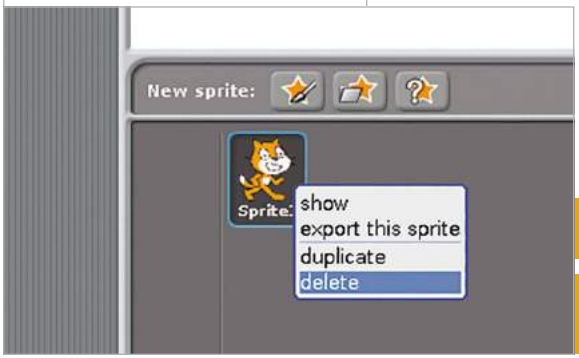
.01



.02



Below: Right-click the sprite in the Sprite List to delete it. Note the buttons to add a sprite above the cat here too



Above: The Operators blocks include the block for picking random numbers, and the blocks for comparing numbers

(Make sure you don't use the repeat block with a number in it). Next, you need to drop a **<** block into the diamond-shaped hole. Click the Operators button above the Blocks Palette to find it. Type -120 into the box on the right. Finally, click the Motion button and drag the y position block into the left box. Now, whatever we put inside the **`repeat until`** bracket will be repeated until the sprite's y position (how far up or down the screen it is), is less than -120. In our game, that means it's missed the trampoline and hit the floor.

>STEP-05

Make the hedgehog move

To make the sprite move, add the two Motion blocks shown in **Listing 4** into the **`repeat until`** block in your script. Click the green flag above the Stage to test it so far. You should see the hedgehog go to the top left, plummet down, and stop when it reaches the bottom.

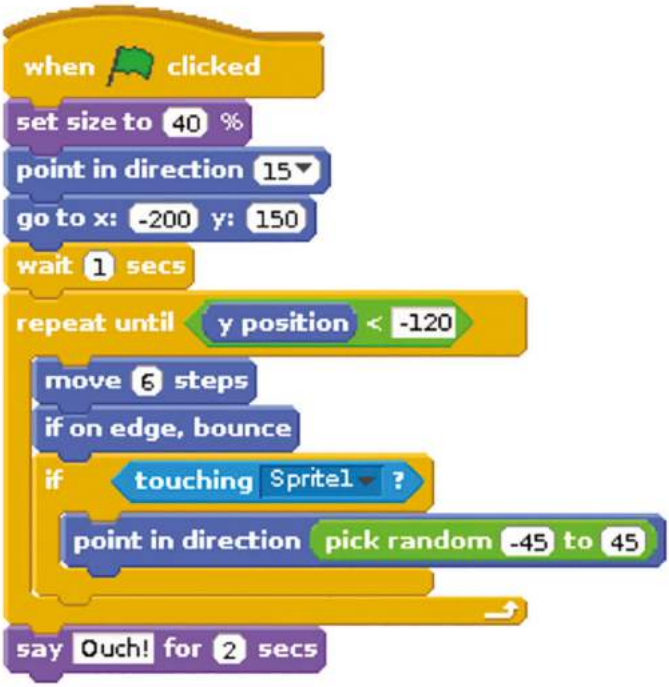
.03

```
when clicked
set size to 40 %
point in direction 15
go to x: -200 y: 150
wait 1 secs
repeat until y position < -120
```

.04

```
when clicked
set size to 40 %
point in direction 15
go to x: -200 y: 150
wait 1 secs
repeat until y position < -120
    move 6 steps
    if on edge, bounce
```

.05



>STEP-06

Make the trampoline bouncy

We need to make the hedgehog bounce back up again if it touches the trampoline. Click the Control block and drag an **`if`** block into your script. Be careful with where you put it: it belongs inside your **`repeat until`** bracket, as shown in **Listing 5**. Click the Sensing button and drag in a **`touching`** block for the diamond hole in your **`if`** block. Click the menu in the **`touching`** block to choose Sprite1 (the trampoline). Inside the bracket of your **`if`** block, put a **`point in direction 90`** Motion block. Instead of putting a number in its hole, this time we'll use **`pick random`** with values of -45 and 45. You'll find it in the Operators section of the Blocks Palette. Now the sprite will point in a random upward direction (between 45 degrees left and 45 degrees right) if it touches the trampoline. Finally, add a **`say`** block at the end of your script, outside all the brackets. This is shown when game ends.

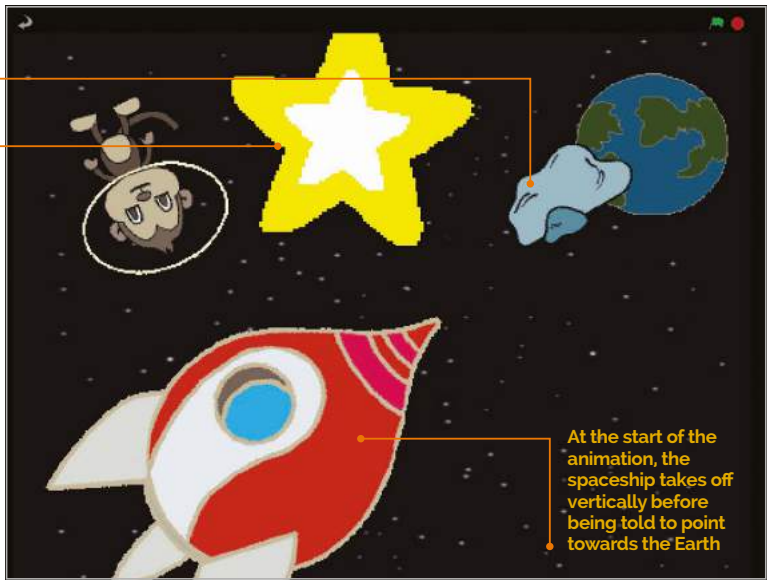
The MagPi ESSENTIALS

[CHAPTER THREE]
LOST IN SPACE

Program your own animation of a spaceship heading for Earth, using a scaling effect to make the ship smaller as it moves into the distance

{code club}

Join your local **Code Club** for more amazing resources like this: **codeclub.org.uk**



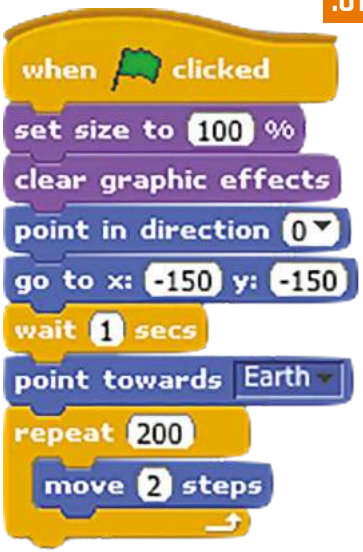
In this chapter, you'll be creating an animation sequence, which, perhaps unexpectedly, involves a rotating space monkey! This project will show you how to move, rotate, and scale sprites. This is something which will also come in very handy for other projects and games. So, start a new Scratch project and get ready to do some animating. If you need any help navigating the Scratch menus, refer to chapter 1.

>STEP-01

Prepare your artwork

After deleting the cat (right-click and Delete), it's time to import a new stage background and sprites. Let's begin by creating our space scene, changing the stage to a field of stars: click Stage in the Sprite List (bottom right), select the Backgrounds tab (top middle), then click Import and navigate to 'stars' in the Nature folder. Since none of the sprites used in this project is in the Scratch 1.4 library, you can download them (**magpi.cc/scratch_art**). First, let's import the Earth and Spaceship sprites: for each, click the star/folder above the Sprite List, then navigate to the folder where you've stored your sprites.

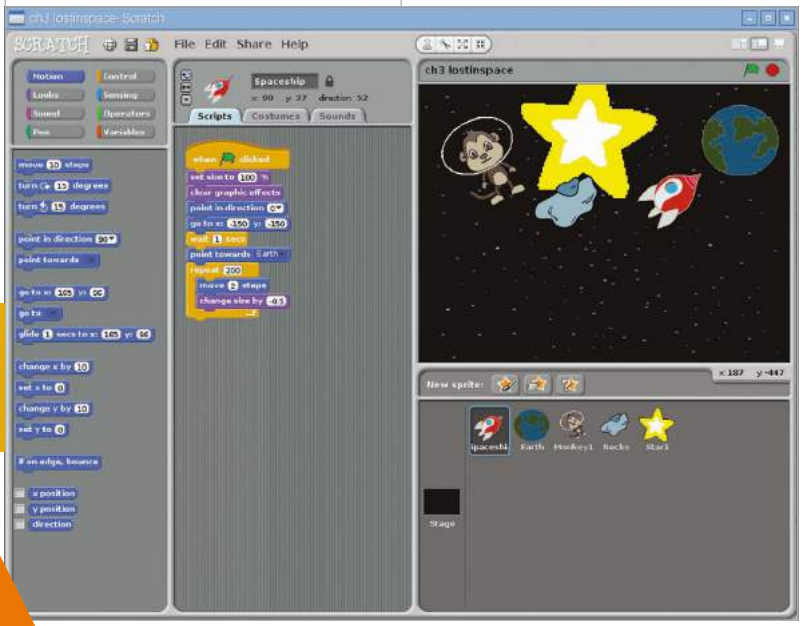
.01



>STEP-02

Move the spaceship

Click the Spaceship sprite in the Sprite List to select it, then click the Scripts tab. **Listing 1** shows the script you need to add to this sprite to make it move. First, we point it upwards (`point in direction 0`) and tell it to `go to x: -150 y: -150`, near the bottom-left corner. After waiting one second, we use the handy `point towards` Motion block to point it at our Earth sprite. We then use a `repeat` loop to keep moving it towards Earth, two steps at a time.



Right: The spaceship points towards Earth and is gradually moved and shrunk within a repeat loop

>STEP-03

Scale the ship

To simulate the spaceship moving further away from us, we need to gradually reduce its size as it moves towards Earth. This is easily achieved by adding a single extra block to its existing script. Click the Looks button in the top-left pane and then drag a **`change size by`** block and drop it just below your **`move 2 steps`** block, within the **`repeat`** loop. Change the 10 of the **`change size`** block to -0.5. The code should look like **Listing 2**. Now, try clicking the green flag to see your space rocket hurtle towards Earth, getting smaller all the time.

>STEP-04

Add a space monkey

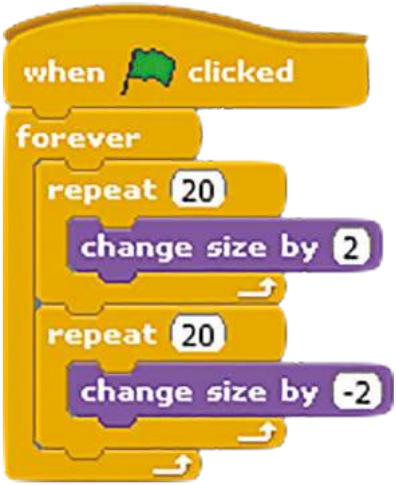
Now let's add a few extra features to our space scene. For a bit of fun, we'll add a floating monkey who's lost in space. Click on the star/folder icon again and navigate to your Lost in Space sprites folder, then select Monkey. As with any sprite, you can adjust its size using the Grow/Shrink sprite icons above the stage. Now let's give our monkey a space helmet! Select it in the Sprite List, then click the Costumes tab and the Edit button. In the Paint Editor, select the Ellipse tool, the outline option (on the right) below the tools, then

.02

.03

Below: In the Paint Editor, draw an ellipse around the monkey's head to give him a space helmet

.04



.05

a yellow colour from the palette. Now draw a yellow ellipse around the monkey's head for a helmet. To make things more interesting, we'll make our monkey spin around by adding the simple looping script in **Listing 3**.

>STEP-05

Bounce and shine

Finally, we'll add a shining star and bouncing rock. Import them both from your Lost In Space sprites folder, then position and scale them on the stage to your liking. For the star, add the code from **Listing 4** (two **`repeat`** loops inside a **`forever`** one) to repeatedly scale it up and down in size. Add the **Listing 5** code to the rock to get it moving, including a special block (as used in chapter 2) to make it bounce off whenever it reaches the edge of the stage.

>STEP-06

Taking it further

Your animation should look pretty cool by now. Try playing around with various parameters to see how they affect the speed, movement, and scaling of the objects. You could also add your own touches, such as using a **`change color effect`** block to give the spaceship a fancy disco-light effect as it moves!

The MagPi ESSENTIALS

[CHAPTER FOUR]
CHATBOT

Nano the cute robot loves to chat. He'll respond to your answers, and he'll even jump up and down if you ask him to...

For this project, you'll be creating your own talking robot which responds to your text input. We'll also alter his expression by switching between different costumes. We'll be using ask commands, if/else blocks, and the join Operator. We'll also create a variable to store the user's name – variables are really handy for storing values to use elsewhere. That's enough chitchat – let's start up a new Scratch project...

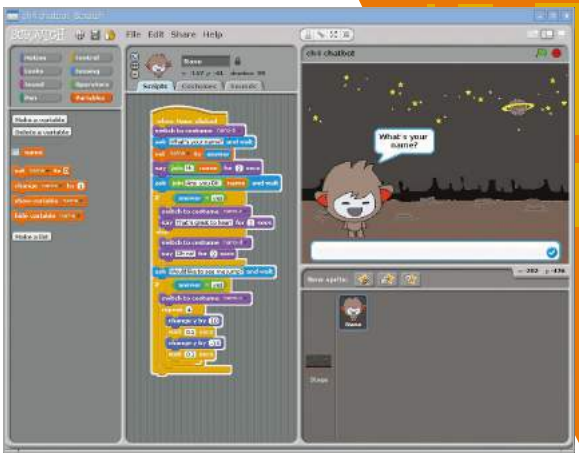
>STEP-01

Prepare your artwork

After deleting that cat by right-clicking on it and selecting Delete, it's time to import a new stage background and our character sprite. Since neither of these is in the Scratch 1.4 library, you can download them (**magpi.cc/scratch_art**). Let's choose a new backdrop: click Stage in the Sprite List (bottom right), select the Backgrounds tab (top middle), then click Import and navigate to the place in the folder where you've stored the downloaded graphics for this project. Next, click the star/folder icon above the Sprite List, then navigate to the same folder and import the Nano sprite. If you click the Costumes tab, you'll notice that Nano has four of them; we'll switch between them to animate our little robot friend.

>STEP-02

Ask for a name



Above: We create a variable to store the user's name and then repeat it within Nano's speech

First, we'll get our robot to ask for the user's name and then use it in a response. With the Nano sprite selected, click the Scripts tab (top middle) and add the code from **Listing 1** (overleaf). Note that instead of using **`when green flag clicked`**, we're starting the program when the Nano sprite is clicked. He then asks for the user's name, which is stored in a variable called **`name`**. First, we need to create the latter: select Variables from the top left, then click 'Make a variable', 'For this sprite only', and enter 'name' in the text field. Untick the **`name`** block to stop it showing on the stage. We can now set **`name`** to **`answer`** (the user's text input) and then add it into Nano's response by using the **`join`** Operator block. Make sure you put a space after 'Hi' to avoid it being joined together with the name.

>STEP-03

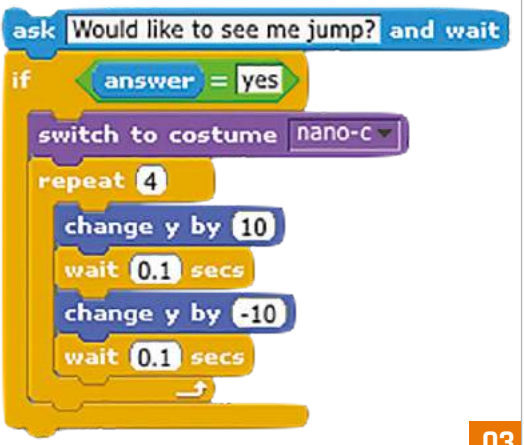
Add a question

Next, we'll add some more blocks from **Listing 2** to the bottom of this script. After saying 'hi' to them, Nano asks the user it they're OK. Again, we use the **`ask`** Sensing block for this, and the **`name`** variable to refer to them by name. We then use an **`if…else`** Control block to determine Nano's response based on the user's input. If it's 'yes' – which we test for using the = Operator – we switch Nano's costume to happy nano-c, using the drop-down box on this Looks block. We also get him to say 'That's great to hear!'

Above: By switching between four costumes, we can alter our character's facial expression

.01

.02



.03

>STEP-04

Else this...

In the **else** part of the **if…else** block, we determine what happens if the user's input isn't 'yes'. In this case, we'll switch Nano's costume to the frowning nano-d and get him to say 'Oh no!' Test out this code with different input to check that it's working as expected. Note that while the user's text input isn't case sensitive, it has to be just 'yes', with nothing added, in order to be recognised as such.

>STEP-05

Jump up and down

Finally, we'll add another question with **ask**, using a standard **if** block to make Nano jump up and down or not; add the blocks from **Listing 3** to the script. We use a **repeat** loop to make Nano move repeatedly up and down for a jumping animation. To make sure he's not frowning from the previous response while doing so, we switch it to nano-c before the **repeat** loop.

>STEP-06

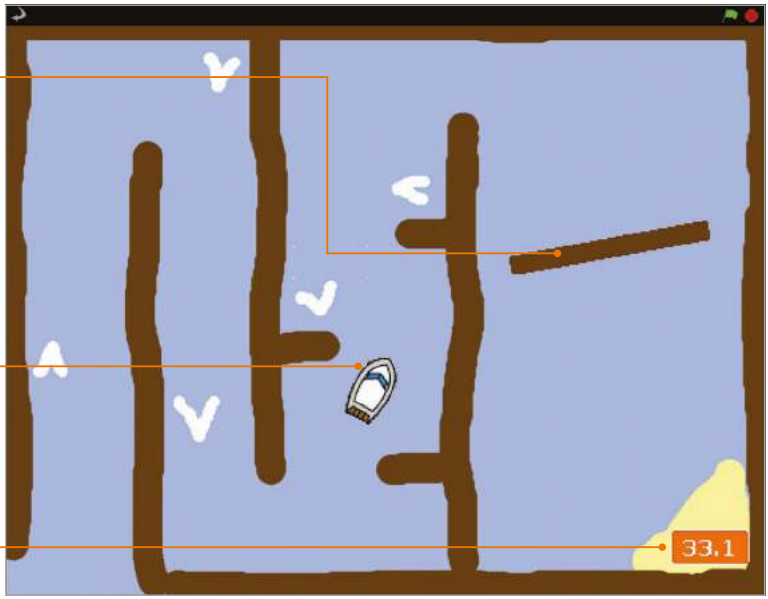
Taking it further

You can alter the example questions or add any extra ones you want, even getting Nano to tell a joke. You could also add extra costumes by copying and editing them in the Paint Editor, or even a design a brand new sprite with various costumes.

The MagPi ESSENTIALS

[CHAPTER FIVE]
BOAT RACE

Create your own boat race game, complete with mouse control, collision detection, and on-screen timer



In this chapter, you'll be making your own arcade game in which the player attempts to guide a boat safely around a maze-like course – including a revolving gate – to the finish in as fast a time as possible. You can even design your own custom course if you like. As well as moving a sprite towards the mouse pointer, this project involves collision detection, using the **`touching color`** Sensing block to determine whether the boat has hit something. Let's dive in and start coding...

>STEP-01

Prepare your artwork

First, delete the cat! You should then import the two sprites, for the boat and gate. Since they're not in the Scratch 1.4 library, you can download them (**magpi.cc/scratch_art**). Just click the star/folder icon above the Sprite List (bottom right), then navigate to the folder where you've stored the downloaded graphics for this project. Import the Boat and Gate sprites. If you aren't designing your own course, you can also download and import our Course backdrop: click Stage in the Sprite List, select the Backgrounds tab (top middle), then click Import and navigate to the folder.

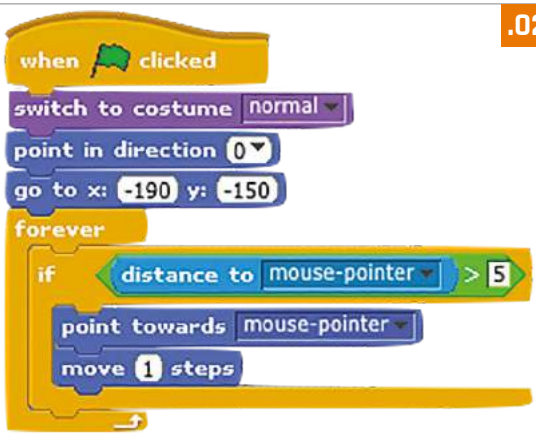
>STEP-02

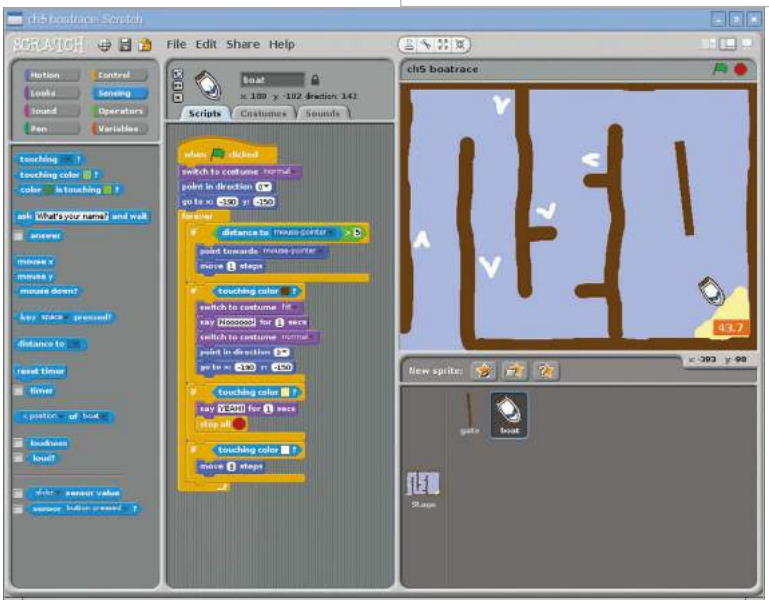
Design a course

You could just edit our course. Alternatively, to create a brand new one, click on the Stage in the Sprite List, then the Backgrounds tab, and Paint. Use the paint bucket tool to fill the canvas with a blue colour for the water. Then use a brown colour – which should be the same as in the Gate sprite – to draw the walls of the course. Use a yellow colour to draw some sand for the finish. Finally, add some white arrows which will act as speed boosters. Once this is done, let's make our Gate sprite rotate by adding the simple code in **Listing 1** to its Scripts area.

.01

.02





Left: We used `touching color` Sensing blocks to detect when the boat has hit a hazard, booster, or the finish

.03

>STEP-03

Controlling the boat

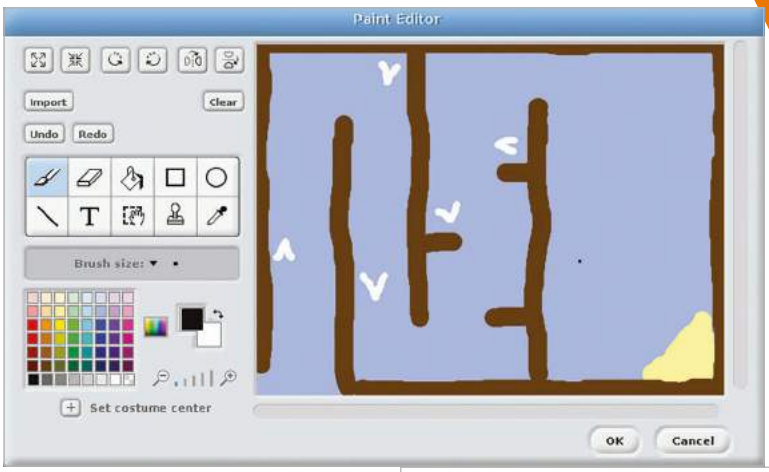
In this game we'll be controlling the boat with a mouse – using the code in **Listing 2** in the Scripts tab of the Boat sprite. To do this, we simply point it towards 'mouse pointer' and move it 1 step at a time, within a **`forever`** loop. To stop it from moving when near the pointer, we put the control code in an if block that only tells it to move if the distance to the pointer is greater than 5. Try out the code and guide the boat: at the moment, it sails straight through barriers.

>STEP-04

Make it crash!

What we need is some collision detection to check whether the boat has hit a hazard. Within your **`forever`** block, add the code from **Listing 3** under your boat control code. Here, we use the **`touching colour`** Sensing block to see if the boat has hit anything brown: click the colour square to get a dropper tool, then click on a brown part of the course. When it crashes, we switch the boat's costume, say 'Noooooo!', then place it back at the start point (in its normal costume).

Let's add two more **`if touching color`** blocks, shown in **Listing 4**, to our forever loop. The first checks whether the boat has reached the yellow beach, which acts as the finish line, and stops the program. The second detects the white of our booster arrows and moves the boat three steps.



Left: You can edit the course in the Paint Editor or create a brand new one

>STEP-05

Boosters and time

To make our game a bit more exciting, we need a timer. Click the Stage and add the **Listing 5** code to its Scripts area. This sets the time to zero at the start of the game, then gradually increases the **`time`** variable in line with real time – you'll need to create the latter in Variables and make sure it's ticked so that it's shown on the stage.

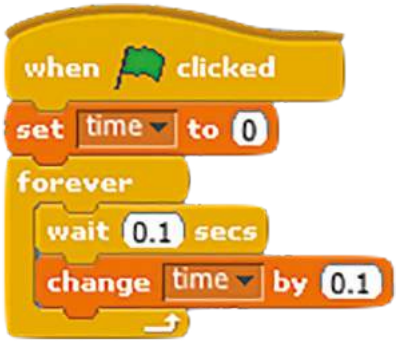
>STEP-06

Taking it further

You could easily add a sound effect for when the boat crashes, using a Sound block. You could even add background music, composing it using Sound blocks with various drums, instruments, and notes. The best time(s) could also be stored in a variable or list.

.04

.05



The MagPi ESSENTIALS

[CHAPTER SIX]

ADA POETRY GENERATOR

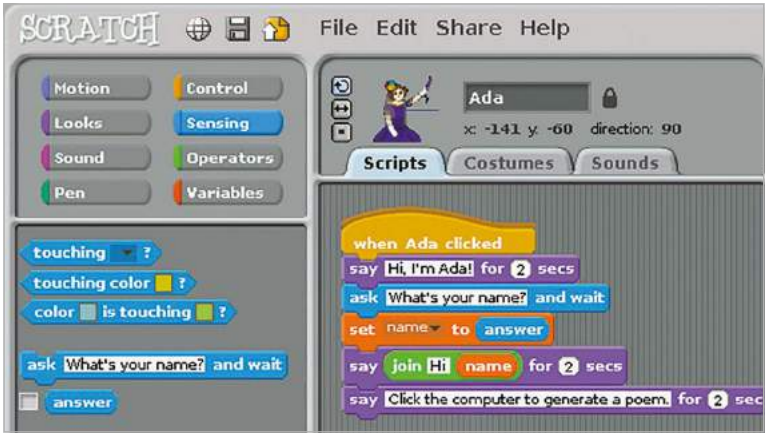
Ada Lovelace unveils the Analytical Engine! This early computer looks a bit primitive, but can generate random poems

In this project, the user first chats to Ada, before clicking on her computer to generate a random poem. To achieve this, we'll be creating and using lists – found in the Variables block category – containing words of a certain type: verbs, nouns, adjectives, and adverbs. We'll then select randomly from these lists to create the poem, which should be different each time. They can be quite amusing.

>STEP-01

Prepare your artwork

After deleting the cat sprite, as usual, you need to import the sprites and backdrop. Since they're not in the Scratch 1.4 library, you can download them (**magpi.cc/scratch_art**). As the Poetry backdrop is so simple – just a grey stripe at the bottom of a white canvas – you could paint it yourself, or just use ours by importing it from the folder where you've stored the downloaded graphics for this project. The same goes for the Banner sprite. Otherwise, import each sprite as usual, by clicking the star/folder icon above the Sprite List.



Right: The script for Ada is similar to that used for Nano in chapter 4, which asks for the user's name

>STEP-02

Ada says hello

Similar to our ChatBot in chapter 4, we'll get our Ada sprite – when clicked – to interact with the user via speech bubbles and text input, using the **say** and **ask** commands. Open the Ada sprite's Scripts tab and type in the code from **Listing 1** (overleaf). As before, you'll need to create a **name** variable: select the Variables block category from the top left, then click 'Make a variable', 'For this sprite only', and enter 'name' in the text field. You should untick the **name** block to stop it showing on the stage. We can now set **name** to **answer** (the user's text input) and then add it into Ada's response by using the **join** Operator block. Make sure you put a space after 'Hi' to avoid it being joined together with the name. After this, we add a block to get Ada to tell the user to click the computer.

>STEP-03

Computer beeps

Click the Computer sprite and select its Scripts tab. This is where we'll add the workings of our poetry generator. To start with, type in the code from **Listing 2** (on page 32). After a block to say 'Here is your poem' and the user's name, we'll use a Sound block to make our computer beep. Our Computer sprite already has the sound for this, or you can record/import a new one in its Sounds tab. We also add a **repeat** loop with two **turn** blocks to make the computer shake.

>STEP-04

Create word lists

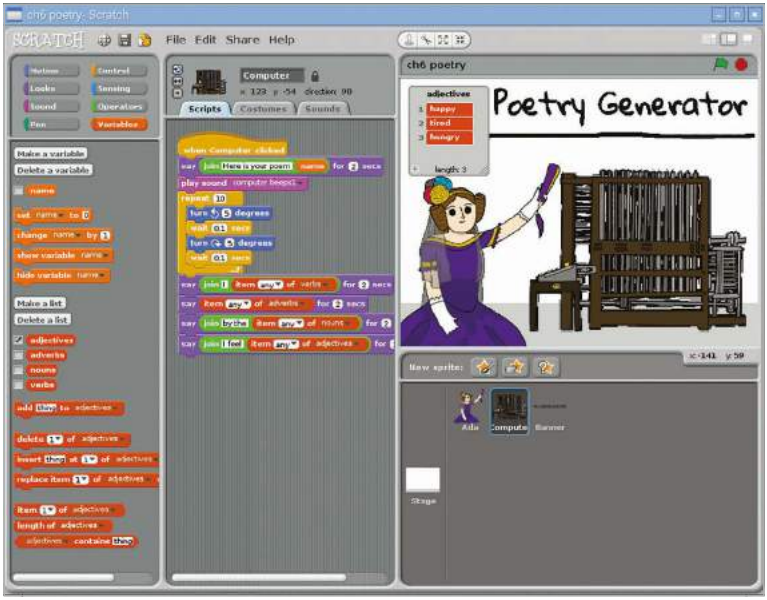
You can't make a poem without words. We'll store ours in four lists: **verbs**, **adverbs**, **nouns**, and **adjectives**. Create each of these in Variables, by clicking the 'Make a list' button, then 'For this sprite only', and typing its name. It will then appear on the stage: to add words to it, click the '+' icon and type them in, one by one. When done, untick this list block to make it vanish from the stage. We used the following words for our lists:

Adjectives: happy, tired, hungry
Adverbs: loudly, silently, endlessly
Nouns: sea, moon, tree
Verbs: laugh, dance, burp

>STEP-05

Poetry in motion

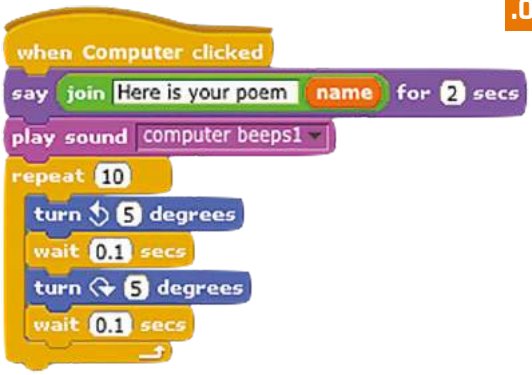
Now we have our word lists, we can use them to generate a random poem each time the computer is clicked by the user. Join the code from



Left: To add words to each list, tick it to make it appear on the stage, then click its '+' icon

.01

.02



Listing 3 to the bottom of your existing script for the Computer sprite. It comprises four **say** blocks, each of which includes an **item of** Variables block; this should have 'any' selected from its drop-down menu, to make a random selection from the list. Test the project out a few times to check that it works properly and generates random poems.

>STEP-06

Taking it further

While we've only created short lists for this example, you could add lots more words to them for greater variation in the random poems created by the computer. More, and differently constructed, **say** blocks can also be added to make poems longer. If you're not keen on blank verse, why not create lists of rhyming words?

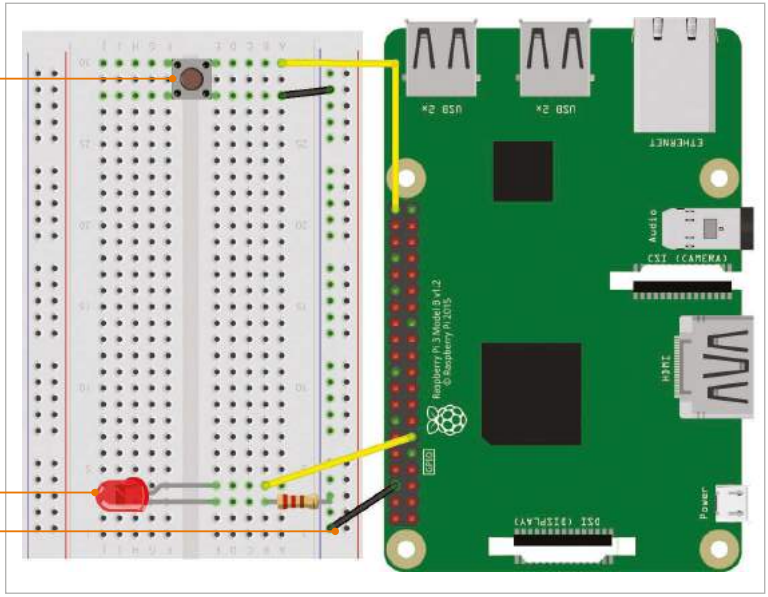
.03

The MagPi ESSENTIALS

[CHAPTER SEVEN]

LIGHT AN LED

Scratch can be used with the Pi's GPIO pins for physical computing projects. Here, we'll hook up a button-activated LED



You'll Need

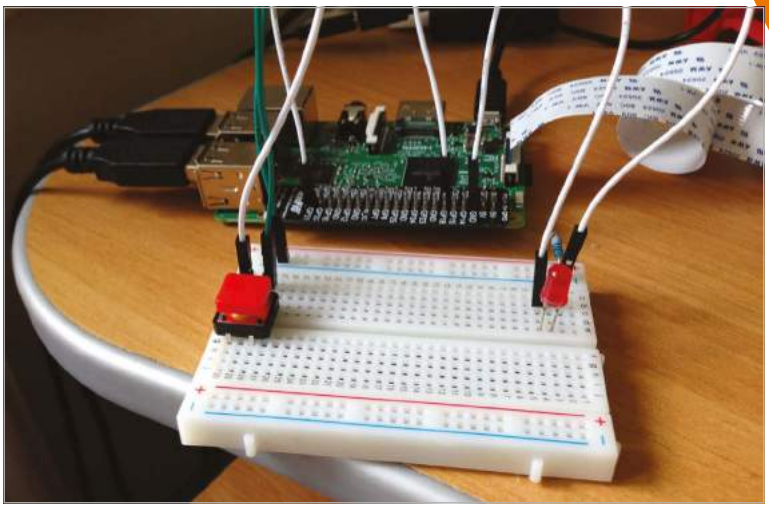
- Solderless breadboard
- LED
- 333Ω resistor
- Push button
- 3× male-to-female jumper wires
- Male-to-male jumper wire

In the latest version of Raspbian Jessie, Scratch features a built-in GPIO server to make it easier to control electronic components or add-on boards. In this first GPIO tutorial, we'll be creating a simple circuit with a button that, when pressed, causes an LED to light up. Take a look at the 'You'll Need' box to see which electronic components are required; you can buy them separately, but they're all in the CamJam EduKit #1 (**magpi.cc/1OcXtim**).

>STEP-01

Connect the LED

It's best to turn the Pi off when building your circuit. The breadboard features numbered columns, each comprising five connected holes. Place your LED's legs in adjacent numbered columns, as shown in the diagram. Note that the shorter leg of the LED is the negative end; in its breadboard column, insert one end of the resistor, then place the other end in the outer row marked '–' (the ground rail). Use a male-to-female jumper wire to connect another hole in that ground rail to a GND pin on the Pi. Finally, use a jumper wire to connect a hole in the column of the LED's longer (positive) leg to GPIO pin 17.



Above: This project is simple to wire up using a solderless breadboard and some jumper wires

>STEP-02

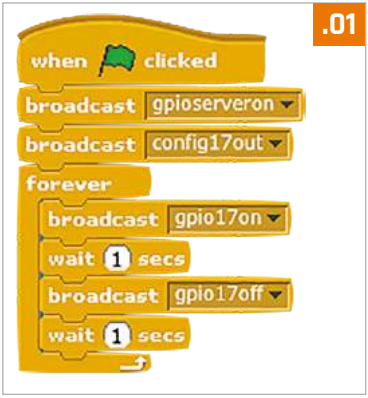
Configure Scratch GPIO

Before we can use the GPIO pins from Scratch, we need to turn its GPIO server on. While this can be done from the Edit menu, instead we'll get our code to activate it. Under a **when green flag clicked** block, add a **broadcast** Control block, click its arrow, select new/edit, and enter **gpioserveron**. We also need to configure GPIO pin 17 as an output pin (to trigger the LED), so add another **broadcast** block and change it to **config17out**.

>STEP-03

Light the LED

We'll now test our circuit by using a loop to make the LED blink. Add a **forever** block to the bottom of your code. Within it, add the following blocks: **broadcast gpio17on**, **wait 1 secs**, **broadcast gpio17off**, and **wait 1 secs**. Now try running the code (**Listing 1**) and your LED should blink on and off continually.



>STEP-04

Connect the button

We can control our LED by adding a push button. Again, we'd advise you to turn the Pi off while connecting new components. Add the push button to the breadboard, with its pins straddling the central groove (as shown in the diagram). Connect a male-to-female jumper wire from one pin's column to GPIO pin 21 on the Pi. Connect a male-to-male jumper from the other pin (on the same side of the groove) to the ground rail you're using for the LED circuit (to share its connection to the GND pin).

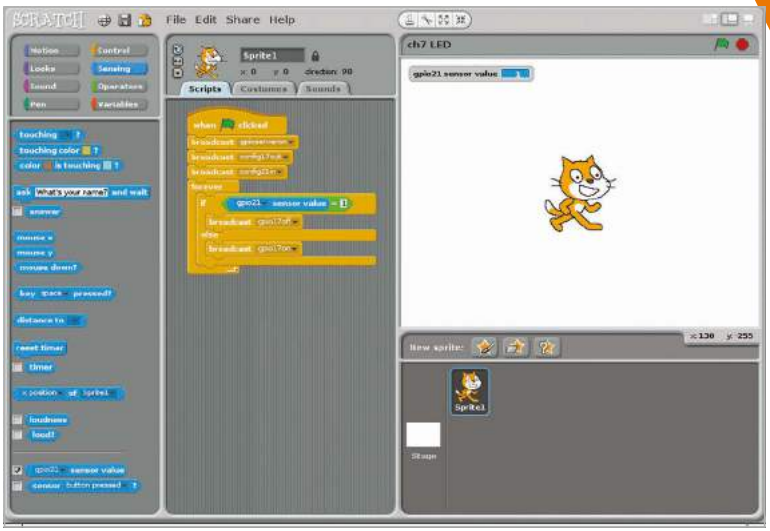
>STEP-05

Configure button

Before Scratch can react to your new button, it needs to be told which pin is its input. Delete the **forever** loop from your blinking LED code, by dragging it out of the area. Add another **broadcast** block with **config21in** to configure GPIO pin 21 as an input – see **Listing 2**. Run and stop the code. Now, click the Sensing category in the top-left pane. Find the **sensor value** block and change it to **gpio21**. Click its tickbox to show its value on the stage: whenever the button is pressed, it should change from 1 to 0.

.02

.03



Left: Ticking the button's `gpio21 sensor value` will show it on the stage, which is handy for testing

>STEP-06

Link to LED

With the button working, it's time to make it trigger the LED. Add the code from **Listing 3** to the end of yours. Again, we're using a **`forever`** block for a continual loop. Inside it we add an **`if…else`** block. In the **`if`** field, we place an **=** Operator block; in its left field, we add **`gpio21 sensor value`**, with 1 in the right field. Underneath, we insert **`broadcast gpio17off`**. This way, when the button isn't pressed, the LED will be off. Under **`else`**, we insert **`broadcast gpio17on`**, to light the LED when the button is pressed. Run the code (as in **Listing 4**), press that button, and watch your LED! In the next chapter, we'll add more LEDs to the circuit to make a pedestrian crossing.

.04

```
when clicked
broadcast gpioserveron
broadcast config17out
broadcast config21in
forever
  if gpio21 sensor value = 1
    broadcast gpio17off
  else
    broadcast gpio17on
```

The MagPi ESSENTIALS

[CHAPTER EIGHT]

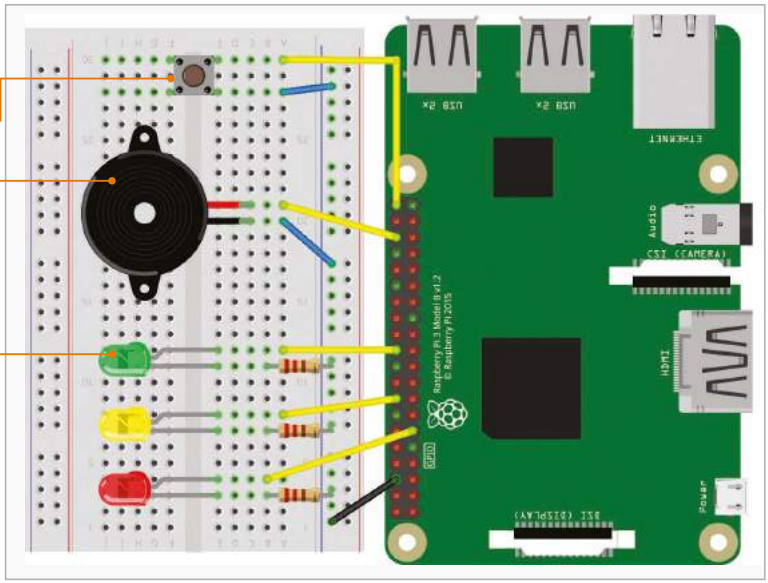
LED TRAFFIC LIGHTS

Following on from the previous chapter, we'll use three LEDs and a push button to make a pedestrian crossing

When the button is pressed, the circuit is broken and Scratch senses a zero value from GPIO pin 21

A piezo buzzer is wired up to the ground rail and GPIO pin 16, for our pedestrian crossing beeps

Each LED is connected to a different GPIO pin, so it can be triggered during the traffic light sequence



You'll Need

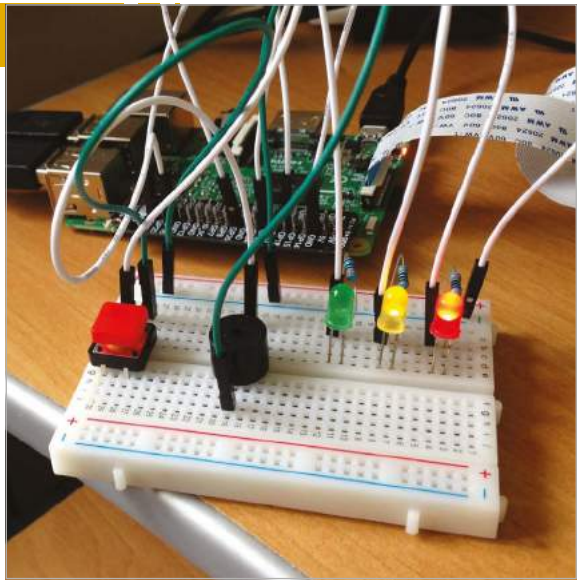
- Solderless breadboard
- 3× LEDs: red, yellow, and green
- 3× 333Ω resistors
- Push button
- Piezo buzzer
- 5× male-to-female jumper wires
- 2× male-to-male jumper wires

In the latest version of Raspbian Jessie, Scratch features a built-in GPIO server to make it easier to control electronic components or add-on boards. In this second GPIO tutorial, we'll create some traffic lights with a pedestrian crossing using LEDs, a push button, and a buzzer. Again, all the components required are in the CamJam EduKit #1 (**magpi.cc/1OcXtim**).

>STEP-01

Connect the LEDs

It's best to turn the Pi off when building your circuit. The breadboard features numbered columns, each comprising five connected holes. Add the LEDs to it, as shown in the diagram. If you've just finished chapter 7, you can leave those components, including the red LED, in place. As before, the shorter (negative) leg of each LED should be connected via a resistor to the '–' row (common ground rail), which is wired to a GND pin on the Pi. Each LED's longer (positive) leg should be connected to the respective GPIO pin via a male-to-female jumper cable.



Above: While there's quite a jumble of wires, it's relatively easy to connect all the components

>STEP-02

Configure Scratch GPIO

First, we need to turn on Scratch's GPIO server. Under a **when green flag clicked** block, add a **broadcast** Control block, click its arrow, select new/edit, and enter **gpioserveron**. We also need to configure our LEDs' GPIO pins as outputs, so add three more **broadcast** blocks and change them to **config17out**, **config23out**, and **config25out** respectively. While we're at it, we'll configure the pins for the buzzer (**config16out**) and button (**config21in**) we'll use later – your code should look like **Listing 1**.

>STEP-03

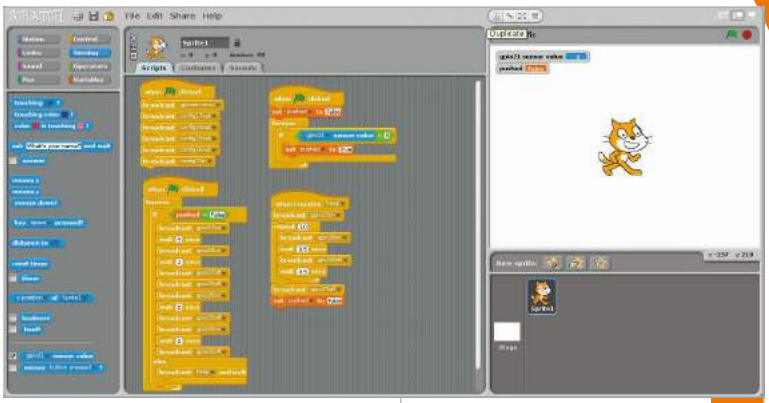
Traffic light sequence

We'll now test our circuit by creating a traffic light sequence: red, red/amber, green, amber. Add the code from **Listing 2**. Here, within a **forever** block, are blocks to turn the LEDs on and off in the correct sequence, waiting a few seconds between each change. Try running it to check that all the LEDs are connected correctly and working.

>STEP-04

Connect the button

For our pedestrian crossing, we'll need a push button. Again, you can use the one already placed in chapter 7, which straddles the central groove of the breadboard and is connected to the ground rail and GPIO pin 21. We've already configured it as an output in step 2; run and stop that code.



Left: Four pieces of code are used for GPIO configuration, light sequence, button press detection, and buzzer beeping

Now, click Sensing in the top-left pane. Find the **sensor value** block and change it to **gpio21**. Click its tickbox to show its value on the stage: when the button is pressed, it'll change from 1 to 0.

>STEP-05

Stop the lights

We need to get a button press to cause the traffic lights to stay on red for a few seconds. Select Variables from the top-left, then click 'Make a variable' and enter 'pushed' in the text field. Add the code from **Listing 3**, keeping it separate from the rest. Using an **if** block, this sets **pushed** to **True** when the value sensed from GPIO pin 21 is zero, i.e. when the button is pressed. Next, we need to add an **if…else** block to our traffic light sequence code, to stop it when **pushed** is **True**. After moving the light sequence

.02

```
when clicked
forever
  broadcast gpio17on
  wait 5 secs
  broadcast gpio23on
  wait 2 secs
  broadcast gpio17off
  broadcast gpio23off
  broadcast gpio25on
  wait 5 secs
  broadcast gpio25off
  broadcast gpio23on
  wait 2 secs
  broadcast gpio23off
```

.03

```
when clicked
set pushed to False
forever
  if gpio21 sensor value = 0
    set pushed to True
```

.04

.05

blocks out of the **forever** block (keeping them in the Scripts area), add in an **if…else** block and put the light sequence blocks back under **if**. In the **if** field, use an **=** Operator block with **pushed** in the left field and 'False' in the right. Under **else**, add a **broadcast and wait** block set to 'beep' – we'll be using this for our buzzer in the next step. Your light sequence code should now resemble **Listing 4**.

>STEP-06

Add a buzzer

Finally, we'll add a piezo buzzer, connected to the ground rail (short leg) and GPIO pin 16 (long leg), to make a beeping noise when it's safe to cross the road. Add the code from **Listing 5** as a separate script. This runs whenever **beep** is broadcast, after the button is pressed and the light sequence ends. It shows a red light and uses a **repeat** loop to turn the buzzer on and off for a beeping sound. Finally, it turns off the red LED and resets the **pushed** variable to **False**. Test out your pedestrian crossing by pressing the button!

The MagPi ESSENTIALS

[CHAPTER NINE]
MULTIPLE-CHOICE QUIZ

Dazzle your friends with your own quiz game, containing hundreds of questions! How many can they get right in 30 seconds?

Click to answer; the answer data comes from a list on Wikipedia

The game runs for 30 seconds before it ends

You'll Need

- LibreOffice – if not installed, open a terminal and type `sudo apt-get install libreoffice`
- List of capitals by size – **wki.pe/List_of_national_capitals_by_population**
- Internet access

Lists are used to remember lots of information, but adding items to them block by block can take a lot of time and Scratch code. In this project, you'll see how you can import (or bring in) large lists from other places, so you can easily make a quiz game with hundreds of questions. As you create this game, you can use your own favourite background and sprites, and arrange them with enough space for the answers to appear. Perhaps you can add your own question list? Anything works, as long as each answer only applies to one question.

>STEP-01

Gather your data

For this game, you'll need two text files: one for the questions and one for the answers. We're going to make a quiz about capital cities, so one file will contain a list of capitals, and the other will contain the countries they are in, in the same order. Start by finding the list of capital cities by population on Wikipedia. Click and drag over the table to highlight it and then press **CTRL+C** to copy it. It's easier if you highlight from the bottom up. Be patient when the screen scrolls!

>STEP-02

Create your question files

Start LibreOffice Calc and paste in the table using **CTRL+V**. Click OK. This might take a minute or two to work. Click above your cities column to highlight it. Press **CTRL+C** to copy the column. Open your text editor, Leafpad, which is in the Accessories menu. Press **CTRL+V** to paste. You should now have a text file containing just capital cities, each one on a new line. If you have a heading at the top (the word 'Capital'), delete it, and remove any blank lines at the end too. Save this file as **`cities.txt`**. Open a new file in Leafpad and repeat the process with the countries column in LibreOffice Calc. This time, save your Leafpad file as **`countries.txt`**.

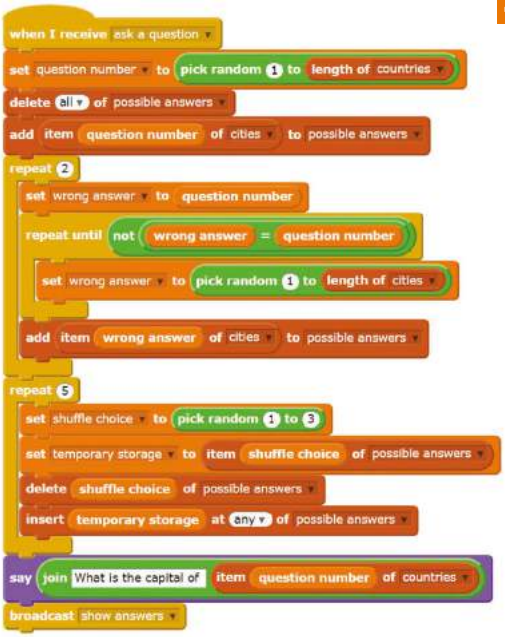
>STEP-03

Importing your data into Scratch

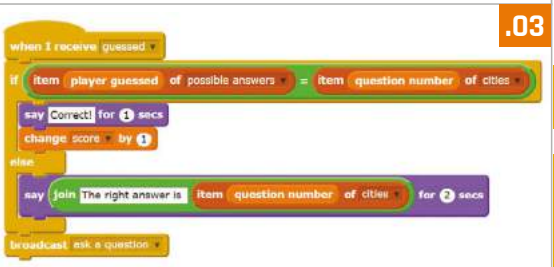
Start Scratch. Click the Variables button and make a list. Call it **`cities`** and make sure it's for all sprites. When the empty list appears on the Stage, right-click it and click **`import`** in the menu. Browse to the files you just created, and double-click your cities text file. The list on the Stage will be filled with the cities from your file. Repeat the process to make a list called **`countries`** and fill it with your countries file. Your list files should be the same length. Right-click the list boxes on the Stage and choose **`hide`**.

.01

.02



.03



>STEP-04

Set up your variables

Through the Variables part of the Blocks Palette, make variables called **`question number`** (used to remember which question/answer pair we're asking), **`score`**, **`shuffle choice`** and **`temporary storage`** (used for shuffling the list of options), and **`wrong answer`** (used when making the list of wrong options). You also need to make a variable called **`player guessed`** to remember which answer the player chooses, and a list called **`possible answers`**. Make all these variables and the list 'For all sprites'.

>STEP-05

Make the main game code

The main game code uses three scripts (**Listings 1–3**). Add them all to the cat sprite. The game uses broadcasts to pass control to the various parts of the program, including on the same sprite. The 'ask a question' section picks a random question number from the list of countries and makes a list of possible answers. It includes the correct answer, and two wrong answers which must be different from the correct answer. The code then shuffles this list to put the answers in a random order, before using a broadcast to make the answer sprites appear and show their answers.

>STEP-06

Make the answer sprites

Import a new sprite to use for showing the answer; we're using Gobo. This sprite has five short scripts (**Listing 4**). Make the variable **`answer choice`**, but click the button to make it 'For this sprite only'. If the game shows all the same answers when you run it, you probably made a mistake here! When you've finished this sprite, right-click it and duplicate it twice. In the copies, change the value of the **`answer choice`** variable at the top to 2 for the first one and 3 for the second one. Happy quizzing!

The MagPi ESSENTIALS

[CHAPTER TEN]
ADD A TITLE SCREEN

To make a professional-looking game, follow these steps to add a title screen with instructions and a fun animation

Black text on hot pink: a timeless background design!

Add an animated sprite to your title screen and use 'say' blocks to tell players how it works

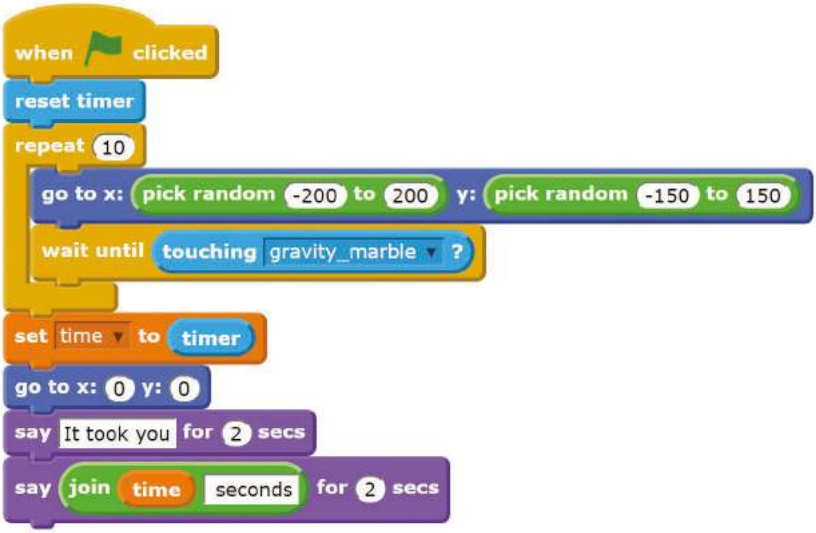
A book has a cover, a film has its credits, and an album has its artwork. Only with the right presentation do these things feel professional and complete. In the same way, a great game starts with a title screen that draws players in and provides instructions. It's especially important if you want to share your game, as you won't be there to explain it when it's played. In this article, you'll see how you can add a title screen to a basic game. The same techniques will work for most simple games, so why not try adding a title screen to your own games, too?

>STEP-01

Write your game

We recommend you try adding a title screen to our example game Cat Catcher before you add one to your own game. To make Cat Catcher, first bring in the sprite Gravity Marble from the Things folder. It comes with some scripts for controlling it with the cursor keys. Add **Listing 1** to your cat sprite. Together, these two sprites make a game where you're challenged to see how quickly you can catch the cat ten times with the marble. We've added the playing field background.

.01



>STEP-02

Create your title screen background

Create a new background image that you'll be using for your game's title screen. Ours is just a bright colour with the game title on it, but you could make something more elaborate if you like. On the background, add the scripts shown in **Listing 2**. They change the background between the title screen and the in-game background, and tell all the sprites to go into 'title screen' mode when the green flag is clicked. Ultimately, this should be the only time you use a `when green flag clicked` script.

.02

.03

>STEP-03

Create your title screen sprite

This is the sprite that will tell the player how to play, and it can be animated too. For our game, we've brought in another cat sprite. Add **Listing 3** to it. There are three parts to this: one part displays the title animation and instructions; another part starts the game when the sprite is clicked; and a third part hides the sprite when the game begins. You'll need to make a variable called **`game status`**, which all sprites will use to tell whether the game is running or the title screen is on. You can add more sprites to your title screen. Include the **`when I receive play game`** script from **Listing 3** to hide them when the game begins. Use a **`when I receive title screen`** script to show them on the title screen.

>STEP-04

Replace your green flag scripts

Now you need to go through your game sprites (the game cat and the marble in our example) and change their scripts so they don't start when the green flag is clicked anymore. For each sprite and each of its scripts, replace the block **`when green flag clicked`** with the block **`when I receive play game`**. Add **Listing 4** to your game sprites to make them hide when the title screen is on, and appear when the game begins. If a sprite shouldn't be there at the start of the game, you can leave out the **`show`** script.

.04

>STEP-05

Replace the forever loops

Some of your in-game sprites might have **`forever`** loops. These will keep running even when the title screen is showing and the sprite is hidden. To avoid this causing unwanted results, replace the **`forever`** block on your in-game sprites with the **`forever if`** block. Give the block the condition **`game status = game`**, using your variable **`game status`** and the **`=`** Operator block. You might also have events that are triggered, such as when there's a key press. To stop these working on the title screen, wrap an **`if`** block around the entire script after the **`when [space] key pressed`** block and give it the condition **`game status = game`**, too.

>STEP-06

Start a new game

When your game finishes, you can show the title screen again by adding a Control block to broadcast **`title screen`**. For example, you could add it to the end of **Listing 1** in our game. Players can once again start a new game from the title screen. That will keep them in the game and encourage them to keep playing until they've got a score they can brag about! You might need to make some other tweaks for your game – each one is different, after all – but following these steps should enable you to add a title screen to most simple games, to make them look more polished.

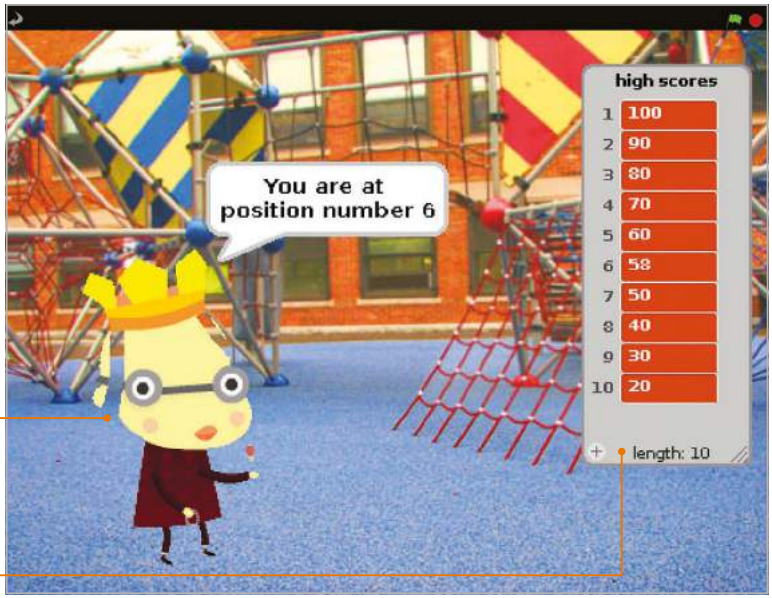
The MagPi ESSENTIALS

[CHAPTER ELEVEN]
ADD A HIGH SCORE TABLE

Keep players coming back for more by keeping a record of the best scores, and telling them how they measure up

The sprite checks the player's score and tells them how they did

Tick the box in the Blocks Palette to see the list and edit its values. No cheating, now!



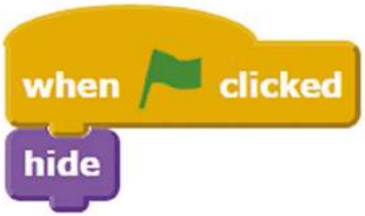
This project features scripts that enable you to create a high score table, and then add new scores to it if they're high enough. There isn't an easy way to display and hide a list from within your program, so the scripts also tell players how they ranked and what the next highest score is, so they know how close they came to beating it. This code will work with most simple games, but you might need to make some changes if your game invites players to play again, or has scripts that continue when the game has ended.

>STEP-01

Make your game

You'll need a game to add this script to – either one of your own, or one that you've programmed from a book or magazine. Try playing the game a few times to work out the likely scores. Some games award a few points, some hundreds, and some thousands. The starting numbers in your high score table should present a challenge to players, but not be completely unachievable. Take care with your own games: if you've spent days playing them in development, they'll be much easier for you than anyone else.

.01



>STEP-02

Add your high score sprite

The scripts for the high score can all go on the same sprite. This sprite will tell players if they got a high score. It could be the main character of your game, the sprite used on the title screen

> This code will work with most simple games, but you might need to make some changes

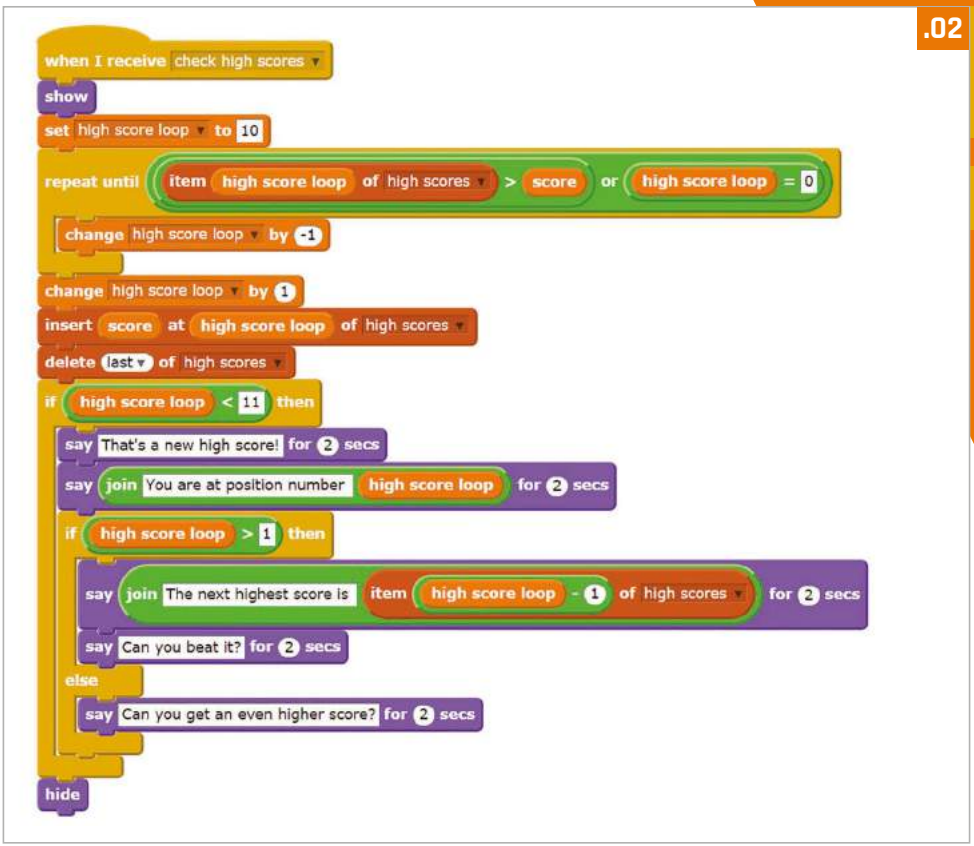
(see the previous chapter), or it could be a new sprite. We've added the sprite **royalperson** for our high score table. You'll find it in the 'people' folder, even though it looks like a dog. It'll be in the way during the game, so add **Listing 1** to hide it when the green flag is clicked.

>STEP-03

Set up your list

Your high score table will be stored in a list. Click the Variables button above the Blocks Palette, click the button to make a list, and call it 'high scores'. In the Blocks Palette, you can click the tickbox beside the list name to show or hide the list on the Stage. This is a handy way to view the whole list, and you can edit the values in it by clicking them and typing on them. The list gets in the way of your game, so we recommend unticking the box.

.02

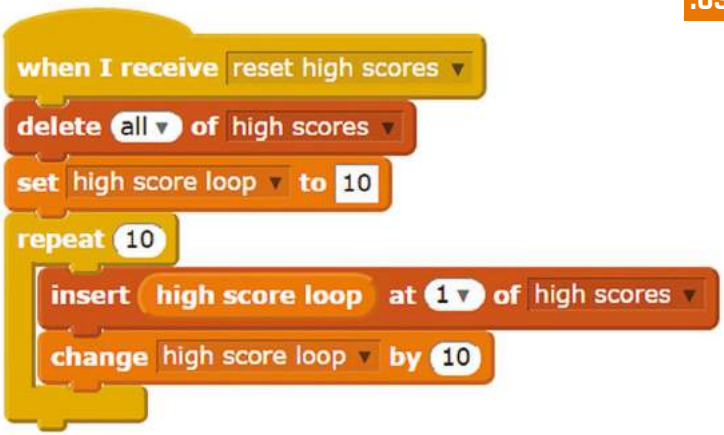


>STEP-04

Set your starting scores

You can type some starting scores into the list on the Stage, but it's better to use a script to generate your high scores. **Listing 2** does this. It runs if it receives the broadcast **`reset high scores`**, but you can also click the script once to reset your scores. To change the lowest score, change the value in the **`set high score loop`** block. To change how much scores go up by, edit the value in the **`change high score loop`** block. Note: the pointed Operator blocks are shown as rounded in our code because of limitations in the Scratchblocks software we've used for laying out code for this book.

.03



>STEP-05

Add your high score code

Listing 3 checks the score and adds it to the high score table in the correct position if it's high enough. It also tells the player how well they did. Add it to your high score sprite. Take care with building the script that goes in the hole of the **repeat until** block. You'll need to drag in blocks in a similar order to this: **or**, **>**, **item 1 of high scores**, **high score loop**, **=**, **high score loop**. When the next highest score is announced, add blocks in the order: **say Hello! for 2 secs**, **join hello world**, **item 1 of high scores**, **-**, **high score loop**.

>STEP-06

Insert it into your game

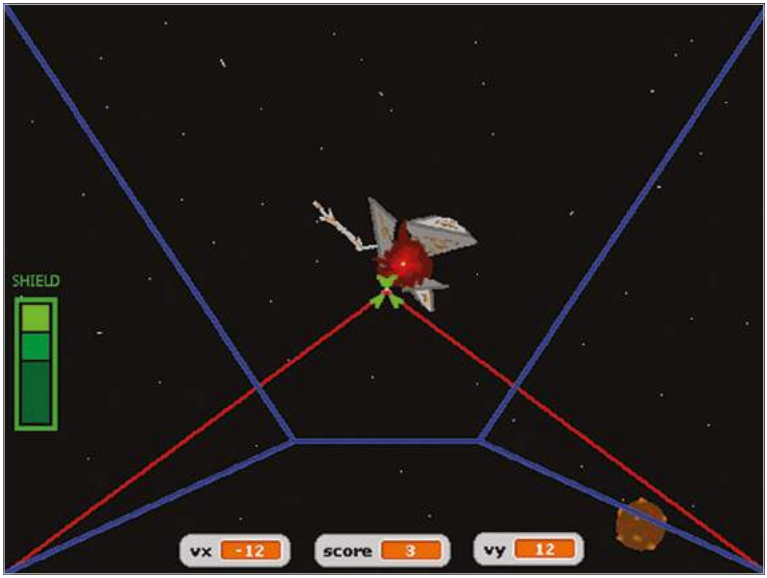
To finish, connect your high score script to your game. If the game doesn't already use the variable **score**, click on Variables and make that variable for all sprites. You want the high score script to run when your game ends, so you need to add some code at that point in your game. Add a block to set **score** to your game's score variable, if you're not already using the variable **score** in the game. Finally, add a block to broadcast **check high scores**. To keep your high scores, simply save your game. When you save a Scratch program, the list values – including your high score table in this case – are saved too.

The MagPi ESSENTIALS

[CHAPTER TWELVE]

BUILD A SPACE SHOOTER

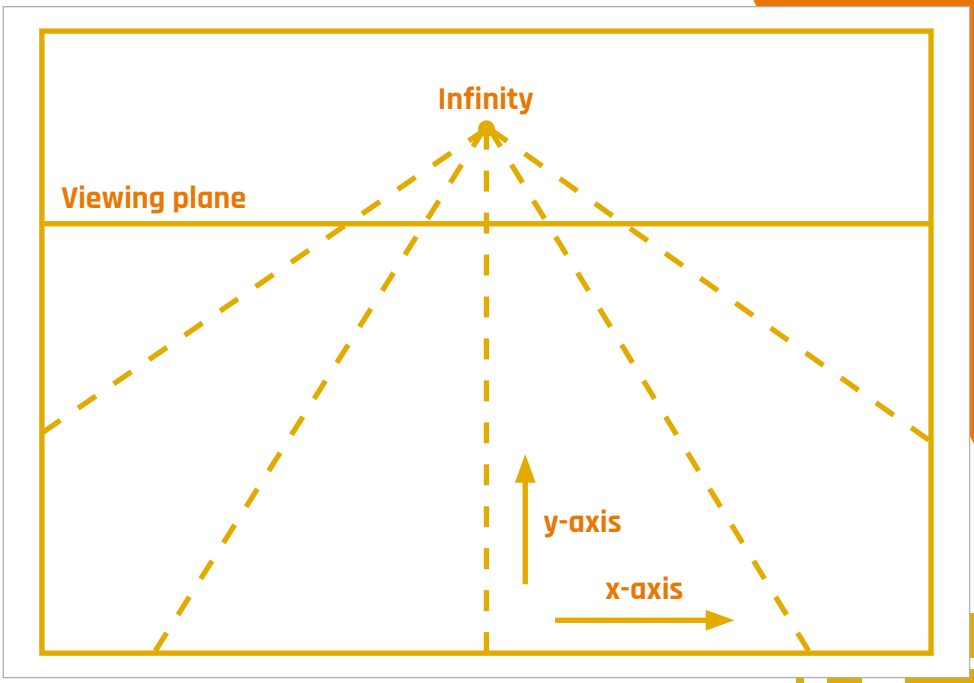
How to create an impressive 3D space shooter, using nothing more than Scratch and some clever coding techniques…



Right: Press the space bar to fire the ship's lasers to destroy debris; if it crashes into you, your shield (green bar) will deplete

Scratch is a great programming language for testing out a range of concepts. As we've seen Scratch programs typically involve controlling one or more sprites on the screen. Computer games where the characters are controlled from a distant view are third-person games. Games can be more exciting when the human player looks through the eyes of the central character in the game, however. This is normally referred to as a first-person game.

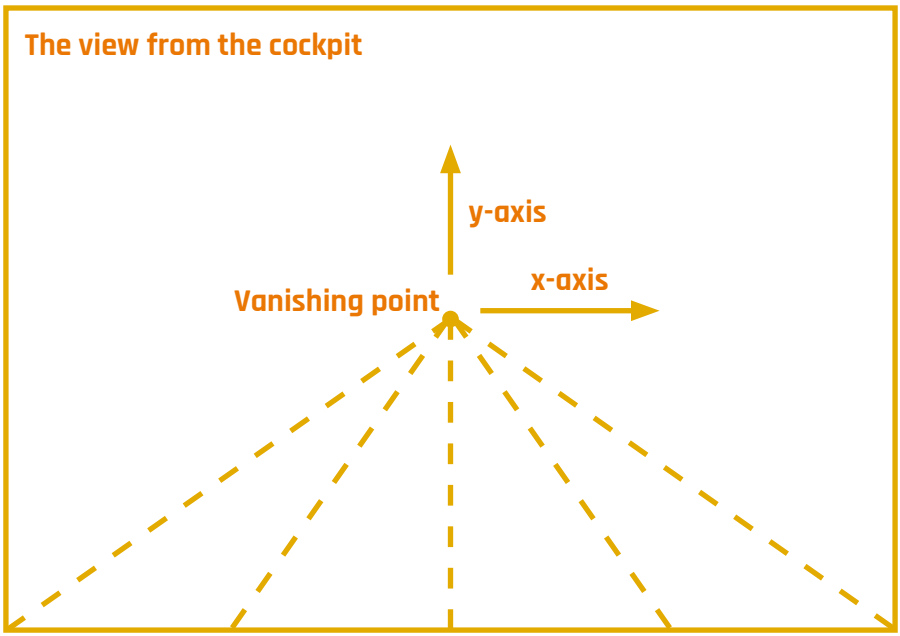
In this article, some of the principles of constructing a first-person game are introduced. The player is the pilot of a spaceship that is drifting through a debris field. The main engine has gone offline, causing the spaceship to drift through the debris at a constant speed. However, the spaceship still has working thrusters on the top, bottom, left, and right of the craft. The main laser system is also operational. The heroic pilot has to shoot through or dodge the debris. A point is awarded each time a piece of debris is destroyed with the ship's lasers. If the debris crashes into the spaceship, then its shield will be damaged. After the shield has been completely broken, the spaceship will explode.



Perspective

In real life, objects that are far away appear to be smaller. One example of this is a set of railway tracks. Looking down railway tracks and into the distance, the tracks appear to become closer together. This can be applied to a computer game, where objects need to be shown as being in the distance. When an object becomes closer to a player, the object should become larger on the screen.

In this game, a one-point perspective is used. This means that distant objects appear to come from the centre of the screen. Rather than draw a lot of very small images at the vanishing point, it's more sensible to assume a viewing plane. The viewing plane corresponds to the distance at which objects become visible. The two diagrams – at the top of this page and overleaf – show the position of the viewing plane, and the vanishing point as it appears on the screen. In the illustration of the viewing plane, the z-axis points from the centre of the screen straight towards the player and is perpendicular to the x-y plane.

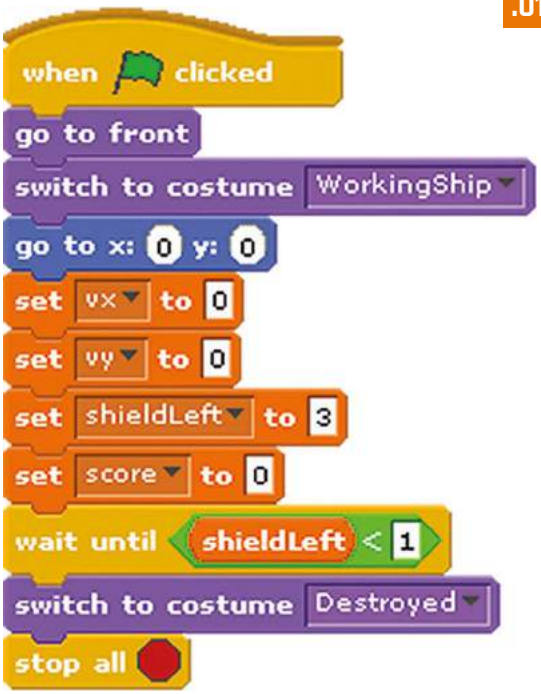


If the spaceship has no velocity along the x-y plane, and an object appears at the viewing plane with a position that's not in the centre of the screen, then the object appears to have a velocity that's proportional to its distance from the centre of the screen. This isn't a real velocity, but is the effect of the perspective used to display the z-axis. This effect can be observed when driving along a straight road: a vehicle that's on the other side of the road, but far in the distance, appears to move to the other side of the road as it approaches.

Spaceship and star field

In the game, the spaceship isn't able to turn. Since the stars in the distance are very far away, they wouldn't appear to move relative to the spaceship. Therefore, a static star field was drawn on the stage background.

.01



The spaceship cockpit and heads-up display should stay in the foreground. This was achieved by creating a sprite that is as big as the screen. When the game starts, the **SpaceShip** sprite is set to be above other sprites (**Listing 1**). Therefore, the cockpit edges are displayed as being in the foreground.

The horizontal and vertical velocity components of the spaceship are stored in the **vx** and **vy** variables. These were created as global variables, since the velocity components affect the motion of other sprites on the screen. The **shieldLeft** variable contains the number of shield points left, and the **score** contains the player's score. The **shieldLeft** variable was created as a global variable, since the other sprites that may hit the cockpit need to be able to change its value; **score** was also created as a global variable, since other sprites need to be able to increment it. The game continues until there are no shield points left. When the game starts, all four global variables are reset to zero and the spaceship is shown to be working as normal. If there are no shield points left, then the ship is shown to be destroyed by changing the costume of the **SpaceShip** sprite. The thrusters on the right, left, bottom and top of the spaceship are controlled by the cursor keys. Since the spaceship is in space, there's no friction to slow down its movement. Therefore, firing the thrusters in one direction will build up the velocity in that direction. To make it easier for the player to see the current status of the game, the values of the **vx**, **vy** and **score** variables were selected to be displayed at the bottom of the screen.

.02

Shield heads-up display

The number of shield points remaining is shown on the left-hand side of the screen. This image is a sprite called **Shield**, which has several costumes that correspond to the different shield states. The different costumes were a copy of the first costume, each with one more green box removed.

When the green flag is pressed, the **Shield** sprite is set to be just below the main cockpit but above the other sprites (**Listing 3**). This means that the shield display stays in the foreground. The script for the **Shield** sprite waits until the number of shield points decreases and then switches to the appropriate costume.

Lasers

The lasers were drawn as another sprite. The size of the **Laser** sprite was carefully matched to the **SpaceShip** sprite by copying the **SpaceShip** costume, to check where the lasers would appear on the screen.

When the green flag is pressed, the **Laser** sprite is set to appear just below the **SpaceShip** sprite (**Listing 4**, overleaf). So it's in the foreground, but not as close as the cockpit. The lasers are fired by pressing the space bar. To make the game a bit harder,

.03

the lasers fire for a second, then recharge for a second. This means that the player should not hold down the space bar, but only fire the lasers when needed. Similar to the **SpaceShip** sprite script, the **Laser** sprite only recognises the space bar when the number of shield points is greater than zero.

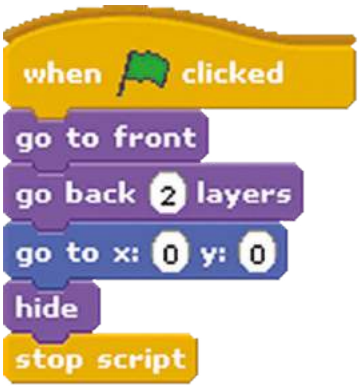
Space debris

Two types of space debris were created: **LavaBall** and **Scrap**. The script for the **LavaBall** sprite (**Listing 5**, overleaf) was copied and modified slightly for the **Scrap** sprite (**Listing 6**) to prevent both sprites appearing at exactly the same time. The two sprites were also given two costumes, to show them as being **normal** or **exploded**.

When the green flag is pressed, the **LavaBall** is placed below the cockpit, shield display, and lasers, then it's hidden from view. The main loop continues while the game is being played. When the **SpaceShip** sprite switches to the **destroyed** costume, it finishes the game by stopping all scripts. This includes the main loops of the space debris sprites.

To show that it's in the distance, the **LavaBall** appears at the viewing plane at 1% of its normal size. To make the game more interesting, its starting position is chosen at random in the x-y plane. Due to the one-point perspective used, objects that are closer to the edge of the screen will quickly disappear from this location. Therefore, objects were chosen to appear within a 100 by 100 box around the centre of the screen. The initial position of the sprite, along the x- and y-axes, is stored in the **initial_x** and **initial_y** variables. Since these variables are only needed for this sprite, they were created as local variables for this sprite only. The initial position components are rescaled to produce an apparent velocity offset associated with the perspective. They are rounded to

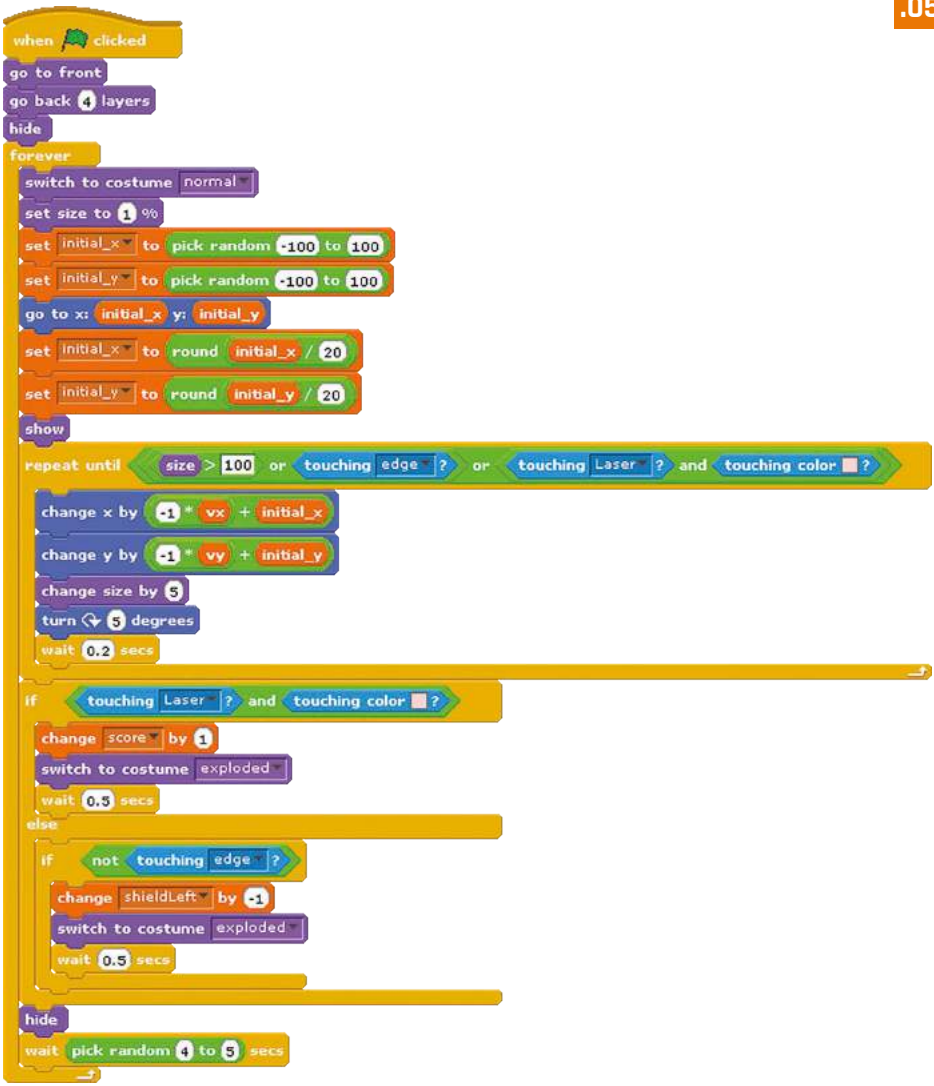
.04



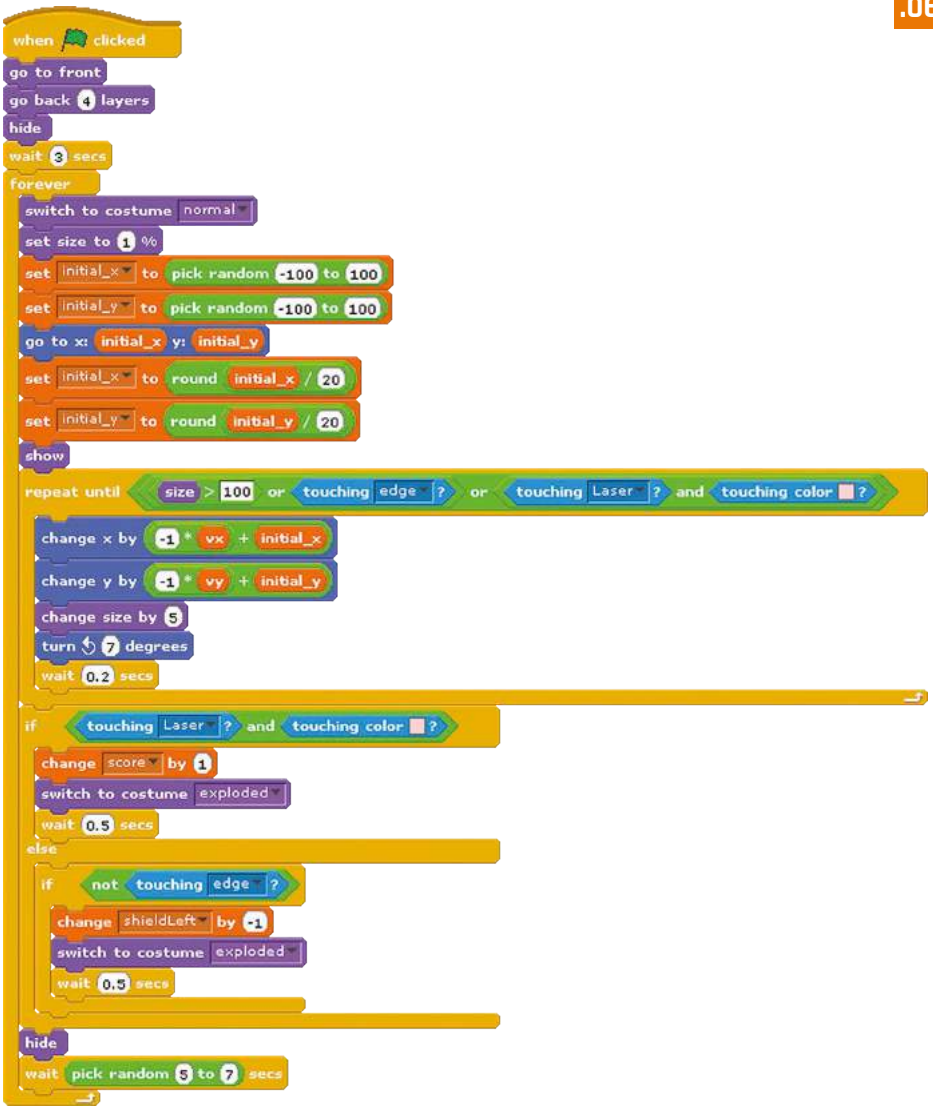
integers, since the sprite moves in numbers of pixels. The sprite is then shown on the screen. Next, the script enters another loop that continues until the sprite is full-size, has touched the edge of the screen, or has been hit by the laser beams. The point where the two laser beams meet was given a pink colour, so that this colour could be used to test if the laser beams had hit the **LavaBall**. The relative velocity of the debris along the z-axis can be raised by increasing the **change size by 5** (5%) command, or by reducing the size of the **wait** within the motion loop.

In this game, the space debris is spinning but is otherwise stationary with respect to the rest of the universe. The spaceship is drifting through the debris field at a constant speed, and starts the game at rest in the x-y plane. When the spaceship thrusters are fired, the spaceship moves along the x-y plane with respect to the universe. However, the game is played from the pilot's point of view, rather than from the point of view of the universe or the space debris. Therefore, when the player's spaceship is moving to the left, the **LavaBall** is shown as moving to the right. If the spaceship moves downwards, then the **LavaBall** moves upwards. This can be demonstrated by looking at a cup on a desk: if the person looking at the cup moves to the left, then the cup moves to the right with respect to the person's line of sight. The motion of the sprite

.05


when clicked
go to front
go back 4 layers
hide
forever
switch to costume normal
set size to 1 %
set initial_x to pick random -100 to 100
set initial_y to pick random -100 to 100
go to x: initial_x y: initial_y
set initial_x to round initial_x / 20
set initial_y to round initial_y / 20
show
repeat until size > 100 or touching edge ? or touching Laser ? and touching color ?
change x by -1 * vx + initial_x
change y by -1 * vy + initial_y
change size by 5
turn 5 degrees
wait 0.2 secs
if touching Laser ? and touching color ?
change score by 1
switch to costume exploded
wait 0.5 secs
else
if not touching edge ?
change shieldLeft by -1
switch to costume exploded
wait 0.5 secs
hide
wait pick random 4 to 5 secs

.06


when clicked
go to front
go back 4 layers
hide
wait 3 secs
forever
switch to costume normal
set size to 1 %
set initial_x to pick random -100 to 100
set initial_y to pick random -100 to 100
go to x: initial_x y: initial_y
set initial_x to round initial_x / 20
set initial_y to round initial_y / 20
show
repeat until size > 100 or touching edge ? or touching Laser ? and touching color ?
change x by -1 * vx + initial_x
change y by -1 * vy + initial_y
change size by 5
turn 7 degrees
wait 0.2 secs
if touching Laser ? and touching color ?
change score by 1
switch to costume exploded
wait 0.5 secs
else
if not touching edge ?
change shieldLeft by -1
switch to costume exploded
wait 0.5 secs
hide
wait pick random 5 to 7 secs

is therefore the sum of the relative velocity and the apparent velocity, due to the object being created at a point on the viewing plane that's not in the centre of the screen.

If the **LavaBall** has been hit by the laser beams, then the score is incremented and the costume is switched to the **exploded** version. The program waits for half a second for the player to view the **exploded** sprite. If the **LavaBall** hasn't been hit by the lasers and it hasn't touched the edge of the screen, then it has hit the spaceship. If the **LavaBall** has hit the spaceship, then the number of shield points is reduced by one and the **LavaBall** costume is switched to the **exploded** version. If the **LavaBall** has missed the spaceship, then it disappears behind the spaceship harmlessly. After these logic conditions, the **LavaBall** sprite is hidden and reappears somewhere else on the screen.

Possible extensions

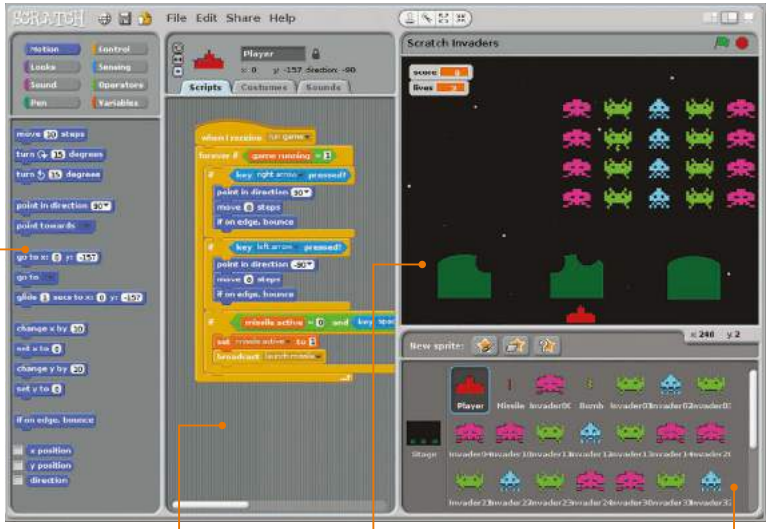
Other features could be added to the game. The spaceship could collect shield tokens or be able to use a wider laser beam to destroy more than one object at once. Alternatively, the principles demonstrated within this program could be used to create a first-person car racing game.

The MagPi ESSENTIALS

[CHAPTER THIRTEEN]
QUICK REFERENCE

To help you get started, here's a handy reference guide to Scratch's interface, GPIO functionality, and all of its code blocks

Blocks Palette: This contains blocks for programming, which you drag into the Scripts Area to add them to your code. There are eight colour-coded categories, selected from the top, each offering a different selection of blocks.



Scripts Area: This is the area where scripts are assembled. It can be accessed from sprites or the Stage, by selecting the Scripts tab. Note that you can create multiple scripts for each sprite. Click the tabs above to switch to costumes or sounds.

Stage: This is where your Scratch creations come to life. Sprites placed here can be resized using the grow and shrink icons above. Click the green flag to start the project running, and the red circle to stop it. There are also icons to change the view, including full-screen presentation mode.

Sprite List: This contains thumbnails of all your sprites. Click one to select it and edit its scripts, costumes, and sounds. The icons above let you paint a new sprite, import one, or select a random one.

BLOCK SHAPES

Blocks are shaped according to the way in which they are used. There are six main types…

Hat Blocks: These are the Control blocks used to start every script – when the green flag is clicked, a key pressed, sprite clicked, or message received.

Stack Blocks: Shaped like jigsaw pieces to fit under and over others, these perform the main commands within scripts.

C Blocks: Generally resembling the letter C, these Control blocks can be wrapped around others to create loops or check for conditions.

Boolean Blocks: These hexagonal blocks contain conditions that, when invoked, report a value of true or false.

Reporter Blocks: Shaped with rounded edges, these hold values – numbers or strings. They include variables and lists.

Cap Blocks: There are only two of these, found at the bottom of the Control category, used to stop one script or all of them.

SCRATCH GPIO

Scratch on the Pi now features a GPIO server for physical computing

In the latest version of Raspbian Jessie, Scratch features a Raspberry Pi GPIO server to make it easier to drive connected LEDs, buzzers, HATS, and other devices and components. First, you need to turn the server on via the Edit menu or running a **broadcast gpioserveron** block. You can then use **broadcast** blocks to configure and trigger individual GPIO pins, and use pulse-width modulation on pin 18. Other functions include taking a photo with the Camera Module, and obtaining the time and IP address. Certain Pi add-on boards and HATs are also supported, set up by creating an **AddOn** variable and setting it to the respective board name. For full details on this and other GPIO functionality, visit **magpi.cc/1TYX7Jg**.

Below: Using the Pi's GPIO pins

BLOCK REFERENCE GUIDE

A guide to all the blocks in each of the eight colour-coded categories, including tips for their usage...

Motion

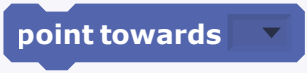
Motion blocks deal with the movement of sprites. They relate mainly to the x and y position and direction of the sprite.

Moves sprite forward by specified number of steps, or backwards (using a minus number). Useful for any project involving movement.

Rotates sprite clockwise by specified number of degrees.

Rotates sprite anticlockwise by specified number of degrees.

Points sprite in the specified direction: 0 = up, 90 = right, 180 = down, -90 = left. Other numbers may also be used.

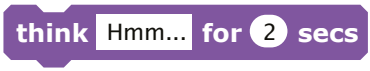
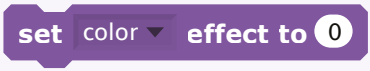
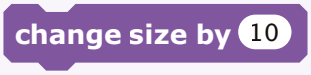
	Points sprite towards mouse pointer or another sprite. Can be used for steering a sprite with the mouse pointer.
	Moves sprite to specified x- and y-position on stage. Useful for resetting its position at the beginning of a project.
	Moves sprite to the location of the mouse pointer or another sprite. Useful for keeping a set of sprites together.
	Moves sprite smoothly to a specified position over specified length of time. One downside is that it pauses the script while the sprite is gliding.
	Changes sprite's x-position by specified amount. Often used in game controls.
	Sets sprite's x-position to specified value. Can be used for horizontal scrolling.
	Changes sprite's y-position by specified amount. Often used in game controls.
	Sets sprite's y-position to specified value. Can be used for vertical scrolling.

if on edge, bounce	Turns sprite in opposite direction when it touches edge of stage. Handy for preventing it partially leaving the screen.
x position	Reports sprite's x-position (ranges from -240 to 240). Tick box to show on stage.
y position	Reports sprite's y-position (ranges from -180 to 180). Tick box to show on stage.
direction	Reports sprite's direction: 0 = up, 90 = right, 180 = down, -90 = left. Tick box to show on stage.

Looks

Looks blocks are used to control the appearance of sprites and the stage. Functionalities include changing costumes and applying graphic effects.

switch to costume costume1	Changes sprite's appearance by switching to different costume. Useful for animation.
next costume	Changes sprite's costume to next costume in the list (if at the end, it jumps back to first costume).
costume#	Reports sprite's current costume number. Tick box to show on stage.
say Hello! for 2 secs	Displays sprite's speech bubble for specified amount of time.

	Displays sprite's speech bubble. (To remove bubble, run this block without any text.)
	Displays sprite's thought bubble for specified amount of time.
	Displays sprite's thought bubble. (To remove bubble, run this block without any text.)
	Changes selected visual effect on a sprite by specified amount. Choose from colour, fisheye, whirl, pixelate, mosaic, brightness, and ghost effects.
	Sets selected visual effect to a given number.
	Clears all graphic effects for a sprite.
	Changes sprite's size by specified amount.
	Sets sprite's size to specified % of original size.
	Reports sprite's size as % of original size. Tick box to show on stage.
	Makes sprite appear on the stage (after being hidden).

hide	Makes sprite disappear from the stage. (Note that when a sprite is hidden, other sprites cannot detect it with a **touching?** block.)
go to front	Moves sprite in front of all other sprites. If it's large enough, it could cover the entire stage.
go back 1 layers	Moves sprite back a specified number of layers, so that it can be hidden behind other sprites.

Sound

These blocks are related to playing various sounds, which can be recorded or imported. 128 built-in MIDI instruments are also available.

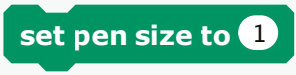
play sound meow	Starts playing selected sound, selected from pull-down menu, and immediately goes on to the next block even as sound is still playing.
play sound meow until done	Plays a sound and waits until it has finished playing before continuing with next block.
stop all sounds	Stops playing all sounds.
play drum 48 for 0.2 beats	Plays selected drum sound for specified number of beats.

	Rests (plays nothing) for specified number of beats.
	Plays selected musical note for specified number of beats. (Clicking the pull-down arrow brings up a two-octave keyboard, but you can enter lower/higher numbers directly.)
	Sets the type of instrument that the sprite uses for **play note** blocks. (Each sprite has its own instrument.)
	Changes sprite's sound volume by specified amount. Volume ranges from 0 to 100.
	Reports sprite's sound volume. Tick box to show on stage.
	Changes sprite's tempo by specified amount (in beats per minute).
	Sets sprite's tempo to specified value in beats per minute.
	Reports sprite's tempo in beats per minute. Tick box to show on stage.

Pen

Pen blocks enable a sprite to draw lines and shapes, including its own 'stamp' image, on the stage when moved.

clear	Clears all pen marks and stamps from the stage.
pen down	Puts down sprite's pen, so it will draw as it moves.
pen up	Pulls up sprite's pen, so it won't draw as it moves.
set pen color to	Sets pen's colour, selected from colour picker. Picking the colour also changes the pen shade.
change pen color by 10	Changes pen's colour by specified amount.
set pen color to 0	Sets pen's colour to specified value (ranging from 0 to 200).
change pen shade by 10	Changes pen's shade (ranging from dark to light) by specified amount.
set pen shade to 50	Sets pen's shade to specified amount. It ranges from 0 (very dark) to 100 (very light). The default is 50, unless set with colour picker.
change pen size by 1	Changes thickness of pen line.



Sets thickness of pen line.

Stamps sprite's image onto the stage.

Control

Control blocks provide functions for looping scripts and only running them if certain conditions are met. The **broadcast** block can be used with the Raspberry Pi's GPIO pins.

Runs the script below once the green flag is clicked to start the project.

Runs script below when specified key is pressed. Useful for player controls in games.

Runs script below when sprite is clicked. Useful for menu buttons/options.

Waits specified number of seconds, then continues with next block. Use it whenever a pause is needed. It's not as accurate as using **timer**.

One of the most commonly used blocks, it runs the blocks inside it over and over, in a never-ending loop.

Runs the blocks inside a specified number of times. Common uses include sprite animation and movement.

Sends a message to all sprites, then continues with the next block without waiting for the triggered scripts. It can also be used to configure and trigger the Raspberry Pi's GPIO pins, and take a photo with the Pi Camera Module.

Sends a message to all sprites, triggering them to do something, and waits until they all finish before continuing with next block.

Runs the script below it when it receives specified broadcast message.

The equivalent of an **`if`** block within a **`forever`** one. Continually checks whether condition is true; whenever it is, it runs the blocks inside.

One of the most widely used blocks. If its condition is true, it runs the blocks inside.

If condition is true, runs the blocks inside the **`if`** portion; if not, runs the blocks inside the **`else`** portion.

Waits until condition is true, then runs the blocks below. Uses include waiting for a sprite to move somewhere, a value to pass a certain amount, or a reply from another script.

Checks to see if condition is false; if so, runs blocks inside and checks condition again. If condition is true, goes on to the blocks that follow.

Stops the script. Handy for disabling scripts, which can be restarted with a broadcast or key press.

Stops all scripts in all sprites. Can be used to end or pause a project.

Sensing

Sensing blocks can be used to detect when one sprite touches another. The **sensor value** block can be used to obtain a Pi GPIO pin's input.

Reports true if sprite is touching specified sprite, edge, or mouse pointer. Useful for collision detection in games.

Reports true if sprite is touching specified colour (selected using eyedropper). Again, handy for collision detection.

Reports true if first colour (within sprite) is touching second colour (in background or another sprite). Both colours are selected using eyedropper.

Asks a question on the screen and stores keyboard input in **answer**. Causes the program to wait until the **ENTER** key is pressed or checkmark is clicked.

Reports keyboard input from most recent use of **ask and wait** (shared by all sprites).

Reports the x-position of mouse pointer.

Reports the y-position of mouse pointer.

	Reports true if mouse button is pressed.
	Reports true if specified key is pressed. Useful for controlling moving objects, such as in games.
	Reports distance from the specified sprite or mouse pointer. Useful in projects that require precision sensing and movement.
	Sets the timer to zero. Handy for when a project or new game level is started.
	Reports the value of the timer in seconds. (The timer is always running.)
	Reports a property or variable of another sprite. Select from: x-position, y-position, direction, costume #, size, and volume. Aids connectivity between sprites in a project.
	Reports the volume (from 1 to 100) of sounds detected by the computer microphone. More precise than **loud?**, it can be used to make sprites react to a certain voice level.
	Reports true if computer microphone detects a sound volume greater than 30 (on scale of 1 to 100).

Reports the value of specified sensor, such as one of the Pi's GPIO pins (or via a connected PicoBoard or LEGO WeDo).

Reports true if specified sensor is pressed. Only used with a connected PicoBoard.

Operators

These provide various mathematical and Boolean operations, along with functions for handling strings.

Adds two numbers.

Subtracts second number from first number.

Multiplies two numbers.

Divides first number by second number.

Picks a random integer within the specified range.

Reports true if first value is less than second.

Reports true if two values are equal.

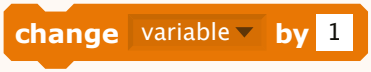
	Reports true if first value is greater than second.
	Reports true if both conditions are true.
	Reports true if either condition is true.
	Reports true if condition is false; reports false if condition is true.
	Concatenates (combines) the two strings.
	Reports the letter at the specified position in a string.
	Reports the number of letters in a string.
	Reports remainder from division of first number by second number.
	Reports closest integer to a number.
	Reports result of selected function (abs, sqrt, sin, cos, tan, asin, acos, atan, ln, log, e^, or 10^) applied to specified number.

Variables

These blocks only appear in the palette once a new variable (changeable value) or list (containing multiple items) is created.

Reports value of the variable. Each created variable has one of these blocks. Tick its box to show it on the stage. Creating a variable named 'AddOn' enables the use of Raspberry Pi add-on boards (see **magpi.cc/1TYX7Jg**).

Sets variable to specified value. Useful for resetting it at the start of a project. This can also be used to set the AddOn variable to use add-on boards such as the Explorer HAT, Pibrella, PiFace, PiGlow, and Sense HAT.



Changes selected variable by specified amount. Uses include altering the speed of an object, level number, or game score.

Shows the selected variable's monitor on the stage.

Hides the selected variable's monitor so it is not visible on the stage.

Reports all the items in the list. (The items are separated by spaces. However, if the items are individual letters or digits, spaces are omitted.)

Adds the specified item to the end of the list. The item can be a number or a string of letters and other characters.

Deletes one or all items from a list. Choosing 'last' deletes the last item in the list. Choosing 'all' deletes everything from the list. Deleting decreases the length of the list.

Inserts an item at the specified position in the list. Choosing 'any' inserts at a random place in the list. Choosing 'last' adds the item to the end of the list. The length of the list increases by 1.

Replaces an item in the list with the specified value. Choosing 'any' replaces a random item in the list. The length of the list does not change.

Reports the item at the specified position in the list. Choosing 'any' reports a random item in the list.

Reports how many items are in the list.

Reports true if the list contains the specified item. Note that the item must match exactly to report true.

OUT NOW IN PRINT

ONLY £3.99

from

raspberrypi.org/magpi

GET THEM DIGITALLY:

Available on the App Store

GET IT ON Google Play


The MagPi
ESSENTIALS
raspberrypi.org/magpi