YOUR OFFICIAL RASPBERRY PI MAGAZINE

The MagPi

Win! 10 RasPi Model A+s Thanks to MODMYPI.COM

100 pages of hacking & making

Issue 39 November 2015

raspberrypi.org/magpi

TIME-LAPSE PHOTOGRAPHY
Make the most of the Pi Camera Module

INTRODUCING GPIO ZERO
Controlling your electronics has never been easier

ASTRO PI UPDATE
T-minus one month and counting!

3D PRINTED GAME BOY
Play your retro games in style

PIXEL ART WITH THE SENSE HAT
Amazing old-school graphics made easy

BUILD A PINGOMETER
How to visualise your internet connection

Everything you need to know to get started today

Also inside:

- MAKE GAMES WITH RASPBERRY PI
- MASTER PIMORONI'S SKYWRITER HAT
- CREATE AMAZING MUSIC WITH SONIC PI
- REVIEWED: SCRATCH, RASPBIAN & MORE

Issue 39 • Nov 2015 • £5.99

9 772051 998001 11

THE ONLY MAGAZINE WRITTEN BY THE COMMUNITY, FOR THE COMMUNITY

Raspberry Pi SWAG STORE

swag.raspberrypi.org

WELCOME TO THE OFFICIAL PI MAGAZINE!

Time is often our enemy. There never seems to be enough of it. I'm often frustrated that, after a long day at work, I rarely get more than an hour to pursue my hobbies before it's time to climb the wooden stairs to Bedfordshire. Since I'm in charge around here, it's only fair that you indulge me this month's cover feature about Raspberry Pi projects I can complete in 30 minutes from start to finish.

It doesn't matter if you're a beginner, a seasoned pro, or you're simply looking for a project that holds the attention of the most demanding offspring; we've got something for you. Get started today with our feature on page 16 – please let us know if you've got a 30-minute project of your own, too.

By the way, do you have an Android or Apple smartphone or tablet? If you do, you might be interested to know that you can now download all 30 original issues of *The MagPi* in our app, as well as get the chance to subscribe to automagically receive every new issue for as little as £20 for a year. Just search for us in your store of choice or find out more on page 90. Once you've downloaded the app, you'll also find the first in a new range of *The MagPi's Essential* e-book range: learn to love the command line with the help of our resident Bash expert, Richard Smedley. You can find out more about his e-book on page 98.

Russell Barnes

THIS MONTH:

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 +44 (0)7904 766523
Features Editor: **Rob Zwetsloot**
Technical Editor: **David Whale**
Sub Editors: **Laura Clay, Phil King, Lorna Lynch**

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

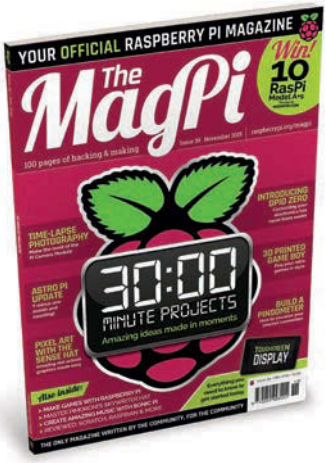
PUBLISHING
For advertising & licensing:
russell@raspberrypi.org +44 (0)7904 766523
Publisher: **Liz Upton**
CEO: **Eben Upton**

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

SUBSCRIPTIONS
Select Publisher Services Ltd
PO Box 6337
Bournemouth
BH1 9EH | **+44 (0)1202 586 848**

CONTRIBUTORS
Sam Aaron, Mike Cook, Gareth Halfacree, Lucy Hattersley, Richard Hayler, Jasper Hayler-Goodall, Phil King, Simon Monk, Matt Richardson, James Singleton, Richard Smedley & Sean Tracey.

This magazine is printed on paper sourced from sustainable forests and the printer operates an environmental management system which has been assessed as conforming to ISO 14001.



Contents

Issue 39 November 2015 raspberrypi.org/magpi

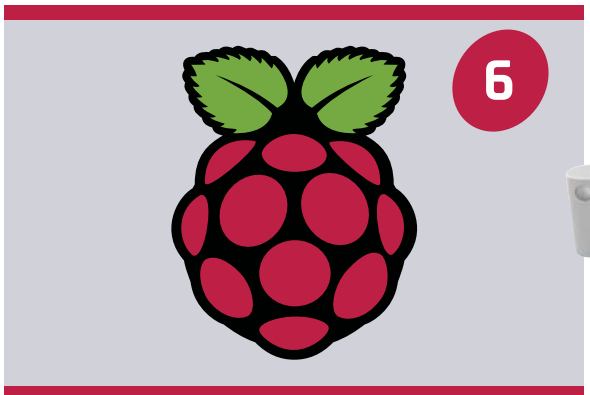
TUTORIALS

COVER FEATURE

30-MINUTE PROJECTS

At a loose end? Grab your toolkit and give one of these quick-fire projects a try

IN THE NEWS



RASPBIAN UPGRADES TO DEBIAN 8 JESSIE

Raspberry Pi's own Simon Long gives us the lowdown on the latest tweaks and changes

A BRIEF HISTORY OF ASTRO PI

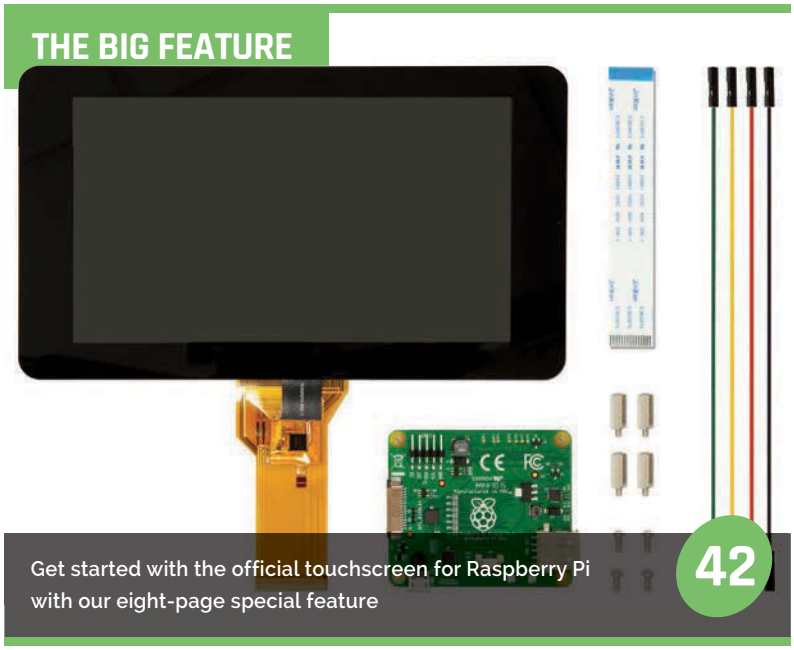
Find out what's been happening with the incredible space mission that'll see two Pis taken to the ISS this December

INTRODUCING GPIO ZERO

Ben Nuttall tells us all about his new easy-to-use package for talking to the GPIO pins on your Raspberry Pi

Contents

THE BIG FEATURE



Get started with the official touchscreen for Raspberry Pi with our eight-page special feature

42

10 RASPBERRY PI Model A+s MUST BE WON!

MODMYPI

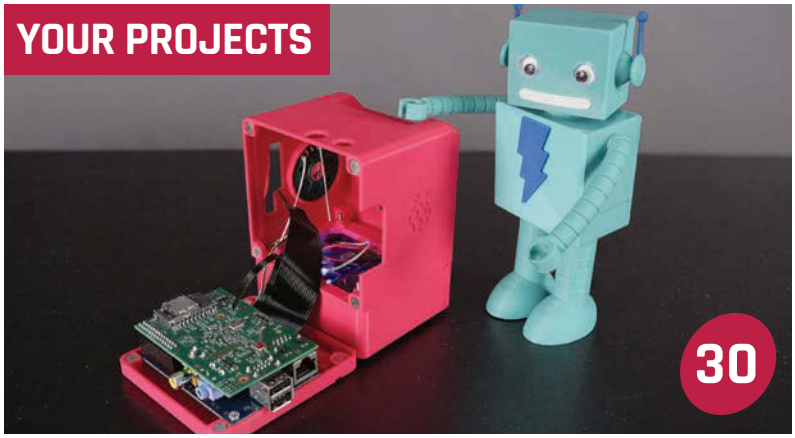
91

A STRONGER FOUNDATION

We speak to the new CEO of The Raspberry Pi Foundation about plans for 2016 and beyond

14

YOUR PROJECTS



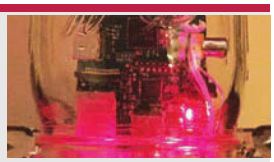
30

MINI MAC PI

Adafruit's Ruiz brothers do it again, this time bringing back the 128kB glory days

Mason Jar 32

Matt Reed shows us how to preserve digital memories in style, with this amazing project

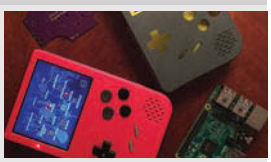


RaftBerry 34

A Raspberry Pi-powered raft, perfect for a picnic on the lake next summer…

Super GameGirl 36

It's smaller than the original Game Boy and plays just about any classic game you can mention



CNC Machine 38

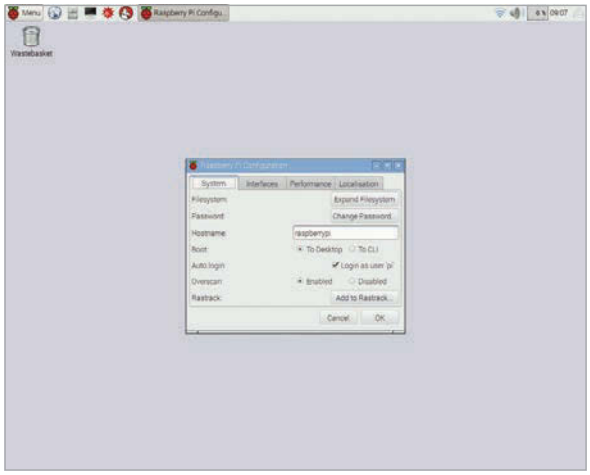
How do you combine a passion for beautiful woodworking with an interest in electronics?

REGULARS

COMMUNITY

REVIEWS

RASPBIAN UPGRADES TO JESSIE



Above The new config menu is much nicer and easier to use

JESSIE SUDOES WHAT WHEEZY SUDON'T

One of the changes that will have a small yet significant effect on Pi users is the ability to access the GPIO pins without having to use sudo on Python or other scripts. Opening a normal IDLE IDE from the menus and trying to run a script that accesses them will automatically do so; this now works from Scratch as well, so even novices can do some physical projects. It's also well timed, now that GPIO Zero is out – check out our news feature on that (page 12-13) for more information on why it makes GPIO easier to use.

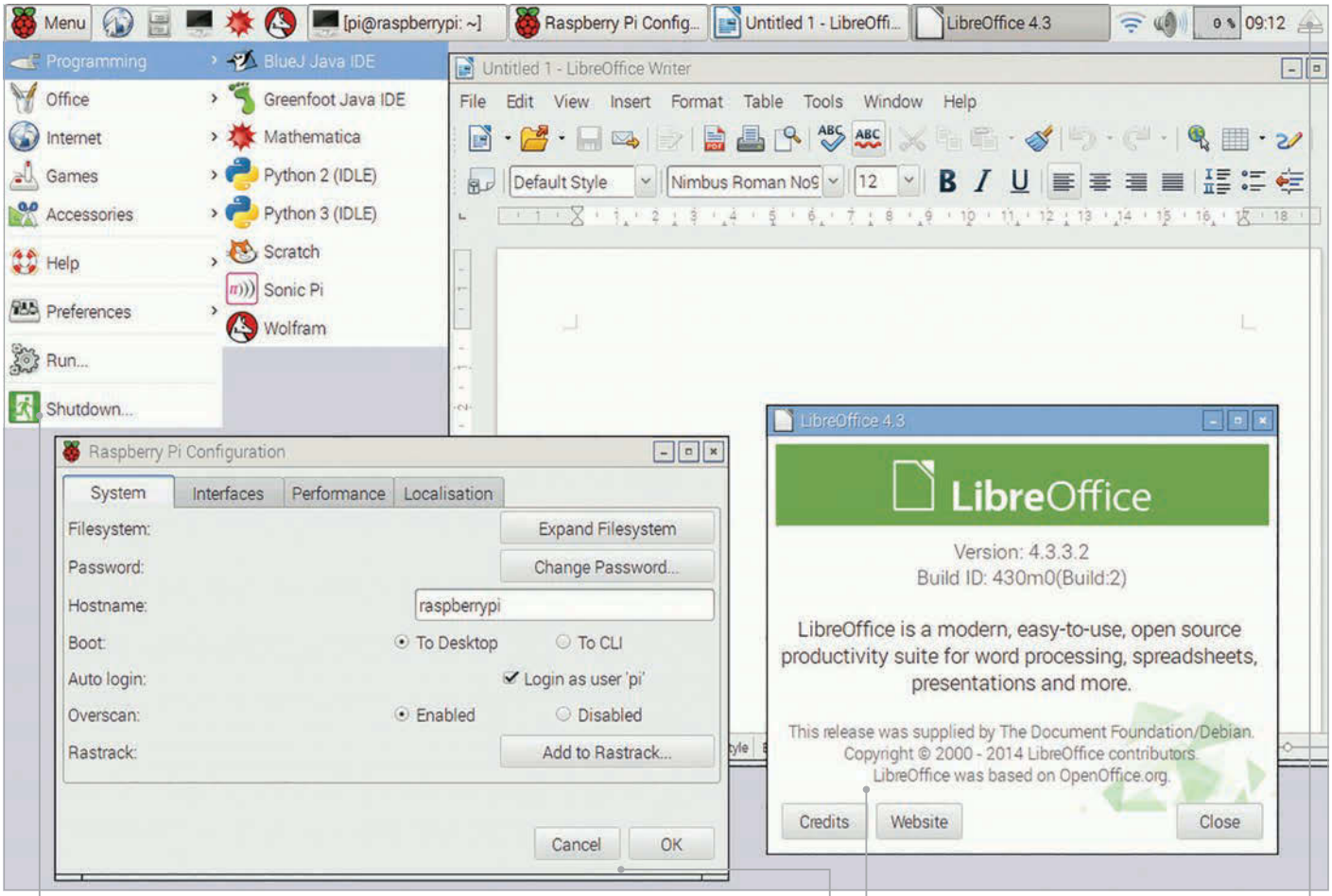
Move from penguin to cowgirl as Raspbian sees some upgrades under the hood and a few more to the interface

For over three years now, Raspberry Pis around the world have been running Raspbian, the official operating system for the Pi. Starting off merely as a version of Debian Linux that was optimised for the Pi, it's since grown to include more and more Pi software, along with an interface overhaul last year and other smaller improvements unique to the Raspberry Pi. What made Raspbian so special was the way it worked more naturally on the ARM chip that powers the Raspberry Pi (known as hard float), rather than some other operating systems that tried to get it working with software (or soft float). Basically, it's easier to work with the hardware naturally than to try and emulate working with it: everything runs a lot smoother on hard float.

For the last three years, Raspbian has been using Debian 7.0 'Wheezy' as its base; at the time, it was the testing branch of Debian, getting rolling updates as they came and allowing for quick updates and upgrades to fix bugs and such. Since then, not only has Debian 7.0 gone stable, the two-year development cycle for this Linux distribution has been and gone, and Wheezy was replaced by the stable version of Debian 8.0, or Jessie, earlier this year.

This brings us to now: Jessie is the newer, shinier version of Debian, with updates to a lot of software along with a few new pieces. The Pi community has been asking for Jessie for a while, and at the very end of September the Raspberry Pi Foundation delivered a new Raspbian based on Debian 8.0.

Hopefully, by the time you're reading this, you've got it installed and set up on your Raspberry Pi. As you may have noticed, there's a bit more to Jessie than just a name



change and a number increment. We spoke to Simon Long, the engineer responsible for user interface design at Raspberry Pi, to talk about all the changes he's been working on.

as a computer in its own right; a computer that could be used as a low-budget replacement to those who need it, whether it's low-income communities or poorer countries. His job is to work

> "There's a bit more to Jessie than just a name change and a number increment"

"I wanted Raspbian to be something I'd want to use," Simon tells us, explaining his reasoning behind the direction of Raspbian over the past year. "I spent a year hacking about at the desktop, which before I started had very little attention paid to it."

Simon and Eben Upton, co-creator of the Raspberry Pi, wanted the Pi to also function

on how the user experience on Raspbian is the best it can be, so that the education side's job can be done much better.

This leads onto the first somewhat controversial change to Raspbian: the switch to desktop by default. "Boot to the command line makes your computer look like a DOS machine from 1985," Simon explains, telling us further that this was a change

Very minor, but nonetheless a great addition, the panel now has an eject USB menu to make sure storage is properly unmounted from Raspbian before you remove it from its physical slot

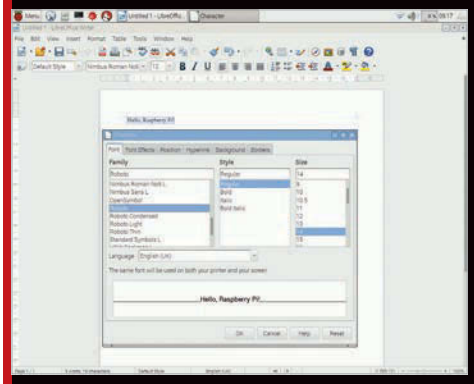
The LibreOffice suite is now installed by default, and works just fine on even the older Raspberry Pi models. This version has been around for a bit, but has seen some upgrades for the Jessie release

The raspi-config settings menu now has its own dedicated graphical tool that performs the same task. It's much neater, cleaner, and easier to use than before

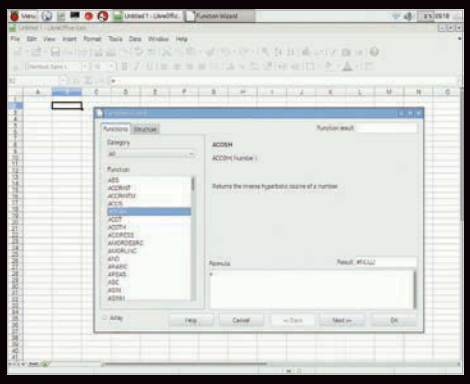
The program menu has some minor aesthetic tweaks but works the same way. The Other category is only hidden now, rather than fully removed, and will appear if a program installs to it

OFFICE PI

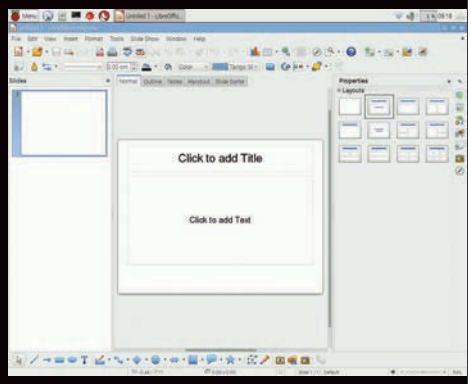
LibreOffice is in Raspbian Jessie by default – but why now?



LibreOffice has been available for a while for Raspbian. "Some work has been done on accelerating this for the Raspberry Pi and a few other places, although that wasn't actually done here. That was done by others in the past. This is a Pi-optimised version, though."



Simon has been working on bundling the Pi version of LibreOffice with a pre-existing plug-in called LibreOffice GTK, which allows LibreOffice to emulate the look of a GTK desktop, the kind of desktop Raspbian uses. It needed some updates made to it first, which Simon worked on.



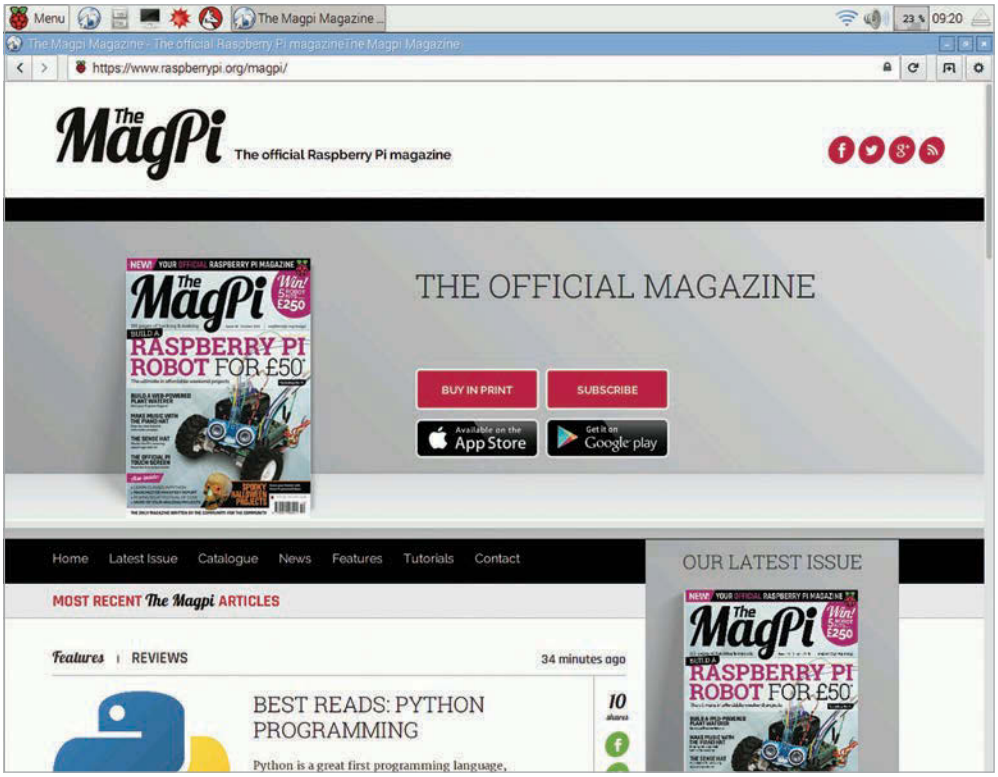
The whole thing came together in time so the entire suite was bundled in Jessie. Write documents with Writer, create and edit spreadsheets in Calc, and put on presentations with Impress. It all works with Microsoft Office files as well, so there's no need to do any conversions.

the educational team were keen on. To them, it shows kids that the Raspberry Pi is a computer like any other, and you can do the same computing tasks with it as you would with your home desktop.

The desktop itself doesn't look a whole lot different to the previous one, but there's been a big change in the technology that runs it.

"The biggest Jessie-related change that I've made is if you look at a window, aesthetics appear a little different," Simon shows us. "What's happened is all of the desktop stuff – the LX panel, the desktop environment – was written in GTK+ 2. Things slowly seem to be moving to GTK+ 3, so I've customised our theme enough that it falls in line with GTK+ 3, and what I've done is made the GTK+ 2 theme match very closely onto the default GTK+ 3 theme. The appearance is tending more towards the newer technology and theme."

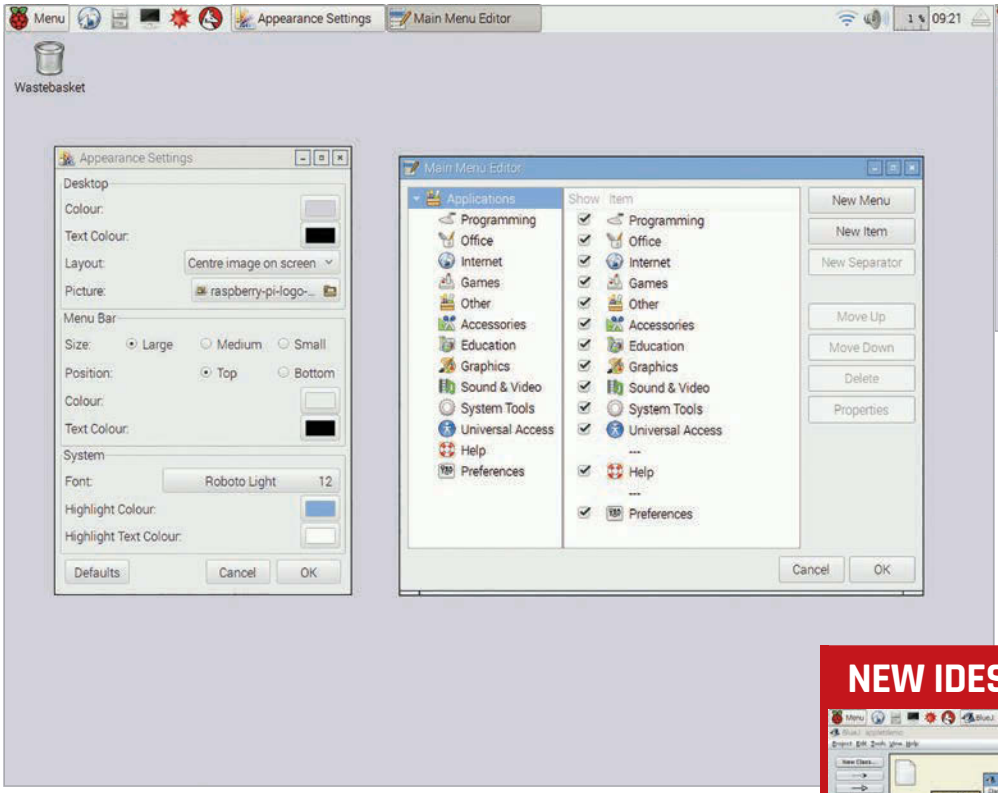
Below Not much has changed with the browser that you can tell, but the tech behind it has been updated

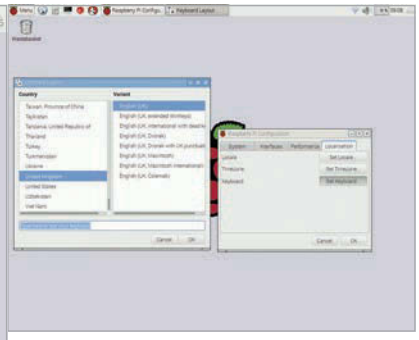


Just browse through the desktop and you'll notice little changes in the colours, spacing and fonts – nothing major, just changes to make it work with GTK+ 3 and give Raspbian a slightly better visual aesthetic overall.

A few new programs come along with that change. First off, there's the now-traditional USB storage device manager on the top panel; from here you can safely eject USB sticks, just like you can on a Mac or Windows PC. "That was an Ubuntu plug-in that I hacked about a bit to work properly," Simon mentions.

The original raspi-config tool – the blue screen you got when you first turned on Raspbian – has now been given its own graphical utility. "In theory, most people shouldn't need to change a huge amount of the stuff that's in here," Simon





Above Simon is proud that the keyboard settings are easier to use, and treat the user as if they know what they're doing

Left The themes have been hacked around to work on both GTK+ 2 and GTK+ 3

explains, but he likes that it's now in a 'reasonably nice, tabbed thing' so that if you do have to access it, you don't always have to Google the right terminal command to get into it. Overclocking options now know what kind of Raspberry Pi you have, keyboard options are much smarter and better organised, and auto-login can be turned on and off for the desktop and CLI individually. Oh, and if you do really miss booting to CLI, that's still an option as well.

The Epiphany web browser has seen a minor update, not that you may notice. "The new version looks exactly like the old version for the time being. Epiphany is moving between WebKit 1 and WebKit 2 and we're about halfway between that. This is still WebKit 1-based; the WebKit 2 bit is still a work in progress as we didn't have time to get it working on Jessie. So the version of Epiphany on here works exactly the same as before: same UI, same look and feel, and should be the same from the user's point of view."

Other little changes have been made here and there. Pressing **ALT** will bring up the button accelerators:

> "The appearance is tending more towards the newer technology and theme"

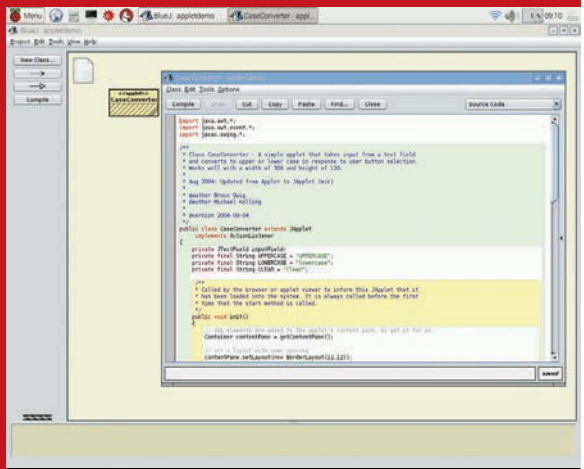
the little lines below items in the toolbar that signify which key to then press to access them without the mouse. The 'Others' category in the program menu also returns, although it's hidden until software that uses it is installed. Pygame Zero, the Python module, is also now included by default.

Jessie's upgrade, then, is all about usability and aesthetics: a newer base operating system with all the advantages that brings, updated software across the board (some with a special Raspberry Pi touch to make it extra special), and tweaks to the way a lot of it looks and feels.

Raspbian Jessie is now available to download from the Raspberry Pi website, in NOOBS, or on its own via **raspberrypi.org/downloads**.

And don't forget to look in the Help category of Software Menu to see a link to our lovely new website!

NEW IDES



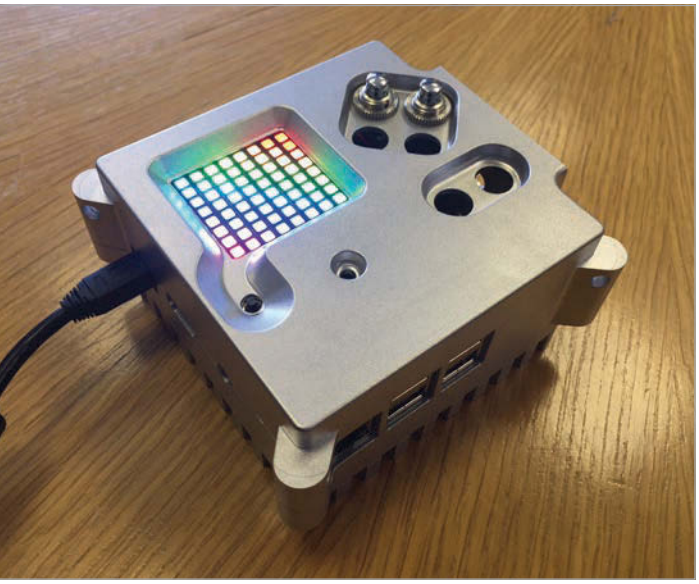
Raspberry Pi has always been the home of Python and Python learning. While versions 2 and 3 of the language are remaining in Jessie (with some tweaked names to make them easier to locate), Raspbian now contains a couple of brand new Java IDEs. These new IDEs, BlueJ and Greenfoot, are products of a collaboration between the University of Kent and Oracle. They offer a different way to create and organise classes that allows for a more visual representation of how you're building code, and is a feasible next step for budding coders to expand beyond Python.

A BRIEF HISTORY OF ASTRO PI

The Astro Pis have been handed over to astronaut **Tim Peake** in preparation for the launch. How did we get here?

On 15 December 2015, British astronaut Tim Peake will blast off to the International Space Station, orbiting the Earth at an altitude of 400km, where he will spend five months performing experiments. Arriving there on an earlier flight – due to cargo overbooking on Tim's – will be two special Raspberry Pis: known as Astro Pis, they are fitted with the Sense HAT sensor expansion board and encased in a special spaceworthy case. Before they've even left the ground, however, the Astro Pis have had an incredible journey to get where they are now.

Below An Astro Pi in its metal case



Your code, in space!

In December 2014, the Raspberry Pi Foundation joined forces with the European Space Agency (ESA), UK Space Agency and UK Space Trade Association to launch the Astro Pi competition: a chance to send British schoolkids' code up to the International Space Station to perform experiments using the Raspberry Pi. This was in conjunction with Principia, a mission to send the first British astronaut up to the ISS to spend five months doing zero- and micro-gravity experiments. The competition period lasted until July 2015, and was introduced by a video from Tim:

"The competition that gives you the chance to devise and code your own program to run on a special version of a Raspberry Pi called an Astro Pi. That's right: your code, in space!"

There were five themes that the entrants were to keep in mind: measurements using the SENSE Hat's environmental sensors (temperature, air pressure, and humidity); use of the movement and motion sensors; satellite imaging and remote sensing; data fusion (i.e. using sensor input to create a conditional response); and space radiation sensing.

"Competition entries will be judged on their creativity, originality, practicality, and usefulness… get those intergalactic hats on, and let's get coding!"

Hardware preparation

It's all well and good for the UK's talented young programmers to provide some great pieces of code, but vital to Astro Pi was making it space certified. Without authorisation from the ESA, it wouldn't be allowed in space and the whole exercise would have been a waste of time. The process for this began in September 2014, long before the competition was even announced.

Everything had to be taken into account. What was the safest way to power the Raspberry Pi? Could permission be granted for the Pi to run off a laptop in certain situations? Was the battery for the real-time clock (RTC) able to survive in space? There were special coatings needed for circuit boards, vibration tests, tests for any noxious fumes, and even a special fondling of the case to make sure it wasn't too sharp. The particulars of each of these and more tests to Astro Pi are included in a lengthy post on the Raspberry Pi blog: **bit.ly/1jCyeYD**.

> It's all well and good for the UK's talented young programmers to provide some great pieces of code, but vital to Astro Pi was making it space certified

Top left Tim Peake is the first British astronaut in 20 years, so it only makes sense he uses a Raspberry Pi in some of his experiments

Above Introducing the competition was a cartoon version of Tim

Left 'Visualisations in Minecraft' is one of the winning entries for the Astro Pi contest

All the testing and poking and refining paid off, though, as on 5 October 2015, over a year since the process began, the Astro Pi was approved for space flight.

Competition results

Shortly after the final deadline for competition entries passed at the beginning of July, the final four winners from the secondary school category were announced, joining the three winning primary school entries. These seven programs each have a codename and do very different things.

'Flags' uses telemetry data on whatever country the ISS is currently orbiting over to display that nation's flag on the Sense HAT's LED display. 'Mission Control' constantly checks and displays the environmental data on the ISS, and will sound an alarm if certain thresholds are met. 'Trees' will take special IR images of the Earth as the ISS orbits, to ascertain the health of trees once the pictures are returned to Earth. 'Reaction Games' is designed to test an astronaut's reactions and alertness after extended periods on board the ISS, using little games. 'Radiation' uses the Camera Module as a radiation detector. 'Sweaty Astronaut' will determine whether someone entering the ISS module will create enough of a rise in humidity to be detected. And finally, 'Minecraft' will visualise movement and environment readings in a Minecraft world.

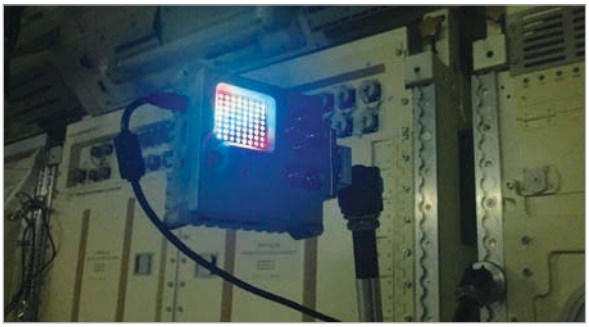
All of these experiments will be run between the two Astro Pis that are sent up to the ISS.

The handover

The final step was laser-etching the cases to have a few snazzy logos, as well as plenty of labels so that each port and sensor could be explained without needing prior knowledge of how the Pi works. Once this was complete, the Astro Pis were sent off to Italy at the end of September for their final tests, which resulted in the certificate being awarded. Their next step is to be loaded onto the Cygnus spacecraft that'll carry them up to the ISS on 3 December.

For now, the Astro Pi's trip is complete. From inception to sitting in a spaceship, it's been a long road, although maybe not as long as the several-hundred-kilometre voyage it will have to make up to space. Its five-month mission: to perform new experiments and find out whether astronauts are really sweaty. To go where no Pi has gone before. Look out for *The MagPi* in January for our first report on the intrepid Pi's adventures throughout the cosmos and low-Earth orbit.

Below An Astro Pi unit is tested in an ISS simulator here on Earth



GPIO ZERO NOW PUBLIC

The GPIO Zero beta was made public in early October, but what is it and why should you be paying attention to it?

Above Ben Nuttall is an education developer advocate at Raspberry Pi, and the creator of GPIO Zero

One of the big new things that's coming out this month is GPIO Zero, a new way to interact with the GPIO ports on your Raspberry Pi. At the time of writing it's now in public beta, but what exactly is GPIO Zero? We spoke to Ben Nuttall, the man behind it.

"GPIO Zero provides a simple interface to everyday GPIO components," Ben tells us. "Looking at 'Hello World' in Java, it's about five, six, seven lines long and there's lots of stuff that you wouldn't really explain; you just have to write it. That's something you don't get with the 'Hello World' of Python, but you do if you're trying to make an LED flash. Rather than import GPIO and set up and set mode and all this sort of thing, it should just be that you have an LED and you turn it on."

Proof of concept

What Ben has ended up creating is a new set of Python classes that automatically do the legwork in preparing the software and hardware to use an electronic component attached to the Pi. In the LED example, you simply import the LED class from gpiozero, create an LED object, providing the GPIO pin, and tell it to turn on, like so:

GPIO ZERO CHEAT SHEET

Here's a rundown of the interfaces you can see in GPIO Zero, and how to use them. For a full list, including add-on boards, visit **bit.ly/1NOAFEq**

INPUTS

BUTTON

To be used with a push button in a circuit. The button can be configured to a 'pull down' circuit, but 'pull up' is the default. Use `wait_for_press()` to wait until the button is pressed to continue with some code, or `wait_for_release()` to run code when the button is let go. There's also `when_pressed`, which causes a GPIO Zero action or custom function to run whenever the button is pressed.

MOTOR

Depending on the motor, you may instead make use of a motor board's libraries, but the tech is available in GPIO Zero. It makes use of `forward()`, `backward()`, and `stop()`.

MOTION SENSOR

PIR motion sensors detect heat differences, so that it can tell when a relatively warm body (e.g. you and your cat, but not Robert Redford in *Sneakers*) is close by. It can be used with two functions: `wait_for_motion()`, which halts the program until motion is detected, or `wait_for_no_motion()` for the opposite effect.

TEMPERATURE SENSOR

Use this function to return the temperature detected by a temperature sensor or thermistor as a value in degrees Celsius.

OUTPUTS

LED

This one you know: it controls LEDs wired up to the GPIO pins. As well as `on()` and `off()`, you can toggle it on or off with `toggle()`, and have it flash on and off with `blink()`.

BUZZER

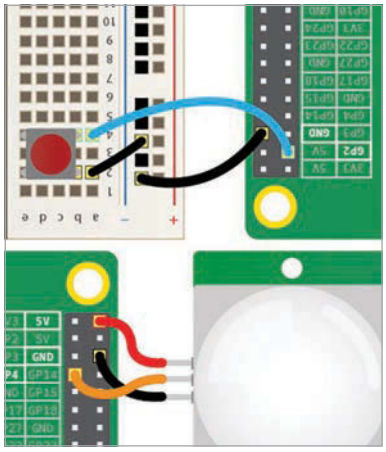
A buzzer component makes a beep when triggered. Like LED, you can use `on()`, `off()`, `toggle()`, and `blink()`. Can you think of a better name for blinking a buzzer? Let Ben know!

RGBLED

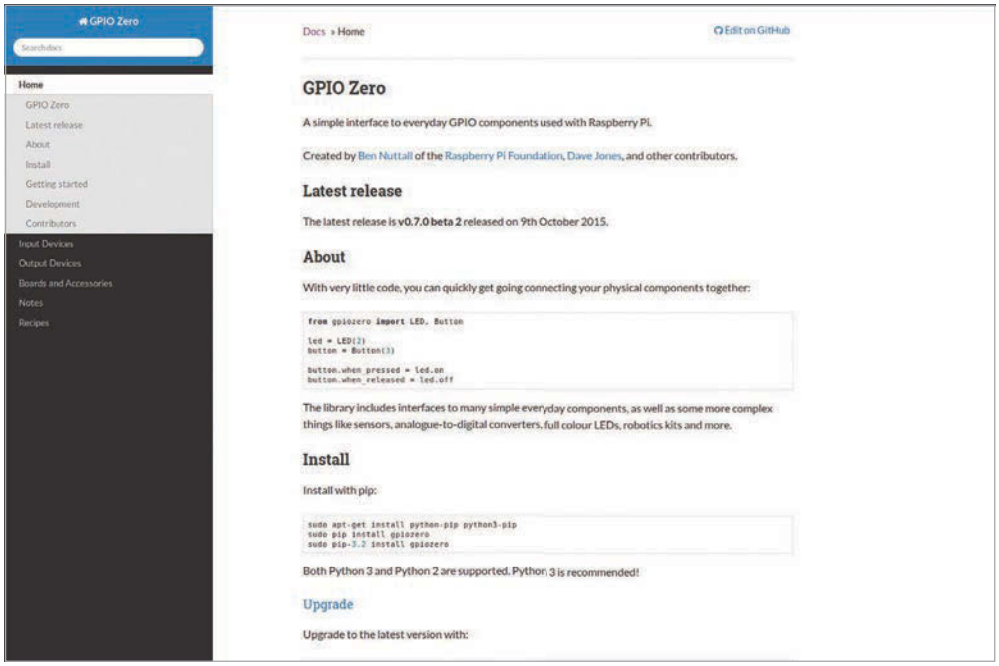
A bit more advanced than the standard LED interface, it requires three pin numbers for red, green, and blue parts of the LED. As well as using the standard `on()`, `off()`, `toggle()` and `blink()`, you can set each LED to a specific brightness to create a unique colour.

LIGHT SENSOR

Similar to the motion sensor, the light sensor uses a light-dependent resistor (LDR), so changes in the amount of light detected result in altered resistance. The methods `wait_for_light()` and `wait_for_dark()` halt the program in the same way the motion sensor does.



Above The documentation is currently being filled with diagrams to easily explain how to wire up components to the Raspberry Pi



Above For comprehensive documentation for all interfaces, including examples and recipes to get started, visit **pythonhosted.org/gpiozero**

```
from gpiozero import LED
led = LED(2)
led.on()
```

Note that the bracketed '2' in this case relates to the GPIO pin numbering scheme.

GPIO Zero also has functions for buttons, motors, motion sensors and more. The idea for this toolset and its development hasn't been the usual year-long task, either.

> It only took about a month before Ben was able to release a public beta of the new library

Humble beginnings

Ben relates the origin story: "I gave a talk at the CamJam at the beginning of September on how to create APIs for Raspberry Pi hardware. Say you've got some hardware and want to create a way to pip install the software and libraries and be able to use it. The same way Pimoroni provide software interfaces to all their products… the talk was basically how to do that. The talk wasn't so much about the code; it was about the methods of uploading to the Python packaging index and that kind of thing. I used trivial examples in my slides – and as we weren't looking at the code, I wasn't trying to teach people how to write in Python – but I was saying imagine you had a board with three LEDs on it and, for instance, you might create an interface that wrapped around the GPIO library meaning that you could turn your LEDs on and off. Looking at that, I thought maybe I should teach teachers how they can create simple interfaces like this, and then I went a step further and said well why teach everybody to do it when it could just be there?"

It only took about a month before Ben was able to release a public beta of the new library, named in a slight homage to Pygame Zero, a minimal boilerplate game framework for education with a similar aim.

Teach a man to fish

While there are about a dozen or so component interfaces provided in GPIO Zero and probably a few more on the way, Ben doesn't really expect there to be functions added constantly to the library:

"I don't really anticipate adding loads and loads of things; it's really just for the everyday stuff. I added in some boards, whatever I had lying around, like Fish Dish which I'm sure is not really used any more, but it's basically in there as a point of concept as a Fish Dish object."

At the time of writing, GPIO Zero can be installed using pip, the Python packaging system, with:

```
sudo pip-3.2 install gpiozero
```

…although you may need to check the GitHub (**bit.ly/1Gkt65k**) to see if you require any extra dependencies. The plan is to have it packaged with the next release of Raspbian Jessie, though, either included as one of the default packages or in the main package system so that it can be installed with apt-get. You can read the full documentation for the library at: **bit.ly/1NOAFEq**.

HELPING HANDS

It would be remiss of us if we didn't mention some of the other people involved with GPIO Zero. Firstly, it's built on top of RPi.GPIO, a library developed by Ben Croston, without which GPIO Zero wouldn't be able to exist in the first place. There is some special code to handle analogue input signals that was created with the help of Martin O'Hanlon. Finally, Dave Jones, who created the Raspberry Pi camera libraries, also helped Ben with the development of GPIO Zero.

PHILIP COLLIGAN

BUILDING A STRONGER FOUNDATION

We chat with the new CEO of the Raspberry Pi Foundation about the organisation's aims and its plans for the next twelve months...

"Our mission is about putting the power of computing and digital making in the hands of people all over the world," says Philip Colligan, CEO of the Raspberry Pi Foundation. "We do that in a number of ways, first and foremost by inventing and selling low-cost, high-powered computers... we don't want price to be a barrier to learning about computing and digital making."

As well as helping people to learn about computers by producing online resources and training teachers, one area upon which he wants the Foundation to focus is to "make computing and digital making more relevant and accessible to people. So we want to work with people on projects that get more kids involved in this field who wouldn't otherwise think it was for them."

Philip tells us that while he's still settling into his new role, he's already impressed by the "open and generous" Pi community and would like it to help shape the development of the Foundation. "The Raspberry Pi computer has been so successful because it's supported by an amazing community of educators, hackers, makers, and businesses who share its goal. I'm keen that people give us their ideas of what we should be doing, give us feedback, and help us figure out where we can add most value in the world."

Pi in space

A major part of Philip's typical day at the Foundation involves meetings with partner organisations: "One of the things that's interesting about our role is we exist in an ecosystem of other people who share our mission, and one of our goals is to try and support them as much as we can."

Of course, one of the most exciting partner projects is the Astro Pi, due to be launched into space in December. "We're ridiculously excited... It's such a privilege to be part of it. We've got a team focused on a whole programme of outreach and educational activities happening between now and December, but we'll be carrying that on till the end of June/July next year. We're planning some big events around the launch, which will be a really special moment."

Two Astro Pis, equipped with sensors and running experiments designed by UK schoolchildren, will be taken aboard the International Space Station by British ESA astronaut Tim Peake. "Obviously, it's super-cool to have a British astronaut," enthuses Philip, "but the fact that a British-made, British-manufactured piece of

SKYCADEMY

Featured last issue, the inaugural Skycademy showed 24 teachers how to launch a Raspberry Pi (equipped with a camera and GPS) into the stratosphere using a high-altitude balloon. While Philip tells us the team hope to run a couple more teacher-training events next year, for now they're focusing on monitoring and partnering with the teacher attendees, who have been given the equipment needed to run their own Skycademy events in their schools. While Skycademy is currently UK-only, Philip reveals, "We're also thinking about how it works in other parts of the world."

Picademy has already trained many teachers and is set to go international next year

technology is going into space and it's at a price that we can give kids the same technology [the Sense HAT] and get them to mirror the experiments that are happening in space... it's amazing." Asked how they could possibly top that, he ponders, "Well, you know, the first Raspberry Pi on the Moon?"

"But seriously," he continues, "there's loads of projects on Earth that we think are really exciting too... I had a great session with [James Bowthorpe] who's running the Hudson River Project, where he is going to sail down the whole length of the river with a Raspberry Pi-powered sensor that assesses the quality of the water... We were talking to them about what would it be like to create a low-cost water-quality tester that you could use all over the world to understand whether or not there's pollution happening in the immediate environment."

Plans for 2016

Philip tells us there's a lot of planning going on at the moment for what the Foundation will be doing next year, including the expansion of the Picademy teacher-training programme. "We want to reach many more teachers with that, so we're thinking about how we partner with different organisations to reach not hundreds [but] thousands of teachers... tens of thousands even." As well as launching Picademy in the USA, the team is in conversation with the Kerala state government in India: "We want to send our teacher training there and we have to think how we do that with a very small team in the UK."

Rolling out the Pi weather station project (as featured in *The MagPi* #32) also requires a great deal of logistical planning. "We are about to send a thousand Pi-powered weather stations to schools all over the planet... one in every country in the world. And that's not trivial, so we're having to learn a lot about how we mail things to different parts of the world... and how we create instructions and lesson plans that teachers, wherever they're based, can use."

Finally, Philip says there will be a major focus on producing more online learning resources in 2016. "Part of what we're doing now is creating projects that people can do at home or in classrooms, giving them a really fun way to learn about computing. We want to create something like a Raspberry Pi curriculum, which any kid who is excited about tinkering away in their bedroom can follow... there will be clear text and lesson guides and practical how-tos that they can work through. A lot of those projects are coming from the community, so we spot teachers or hobbyists creating something and we reach out to them and turn that into an education resource which others can benefit from."

FROZEN PI

While stressing the importance of the Foundation producing online resources to encourage parents to do projects with their children, Philip reveals he's been working on a project with his eight-year-old son, Dylan. Upon being given a Sense HAT with a variety of sensors, including temperature, the first idea Dylan had was to put it in a bucket of ice! "Within a couple of hours, he'd figured out a way, with a battery pack inside a bag, to put the Raspberry Pi into a freezer for three hours. He was learning how you capture the temperature data, write it to a CSV file, and then use that to create a graph... He was just thrilled at the idea that a computer he owns, he's allowed to stick in a freezer and see whether or not he can break it." Fortunately, although it reached -14°C, it was all fine and still working when it came out.

MINUTE PROJECTS

Have a bit of spare time this Saturday? Why not make something quick and fun with your Raspberry Pi?

As a Raspberry Pi owner wanting to do something fun, your choices are near limitless. Whether you're leafing through loads of little projects in the pages of The MagPi, or seeing projects around the internet that look interesting, the only constraints are time and money. Some projects can last months, while others take hours or days, but why not try a project that you could complete in half an hour?

Whether you have a bit of time in the evening to hack around with a Raspberry Pi, or want something quick to do with your kids that'll keep them interested long enough to see the results, we've got a little project just for you. With a mixture of hardware and software projects (and one using spoons), each with its own shopping list, you can get yourself set up in advance to tackle a project in one sitting, in the same amount of time it takes to order a pizza – which, incidentally, is a fine reward for completing your Raspberry Pi project!

START TIMER

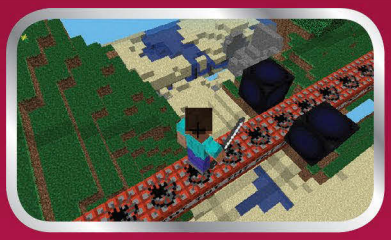
CONTENTS

HIGH-SPEED PHOTOGRAPHY

Create slow-motion video with an adjustable frame rate, and at a much higher resolution, with just a Raspberry Pi, the Camera Module, and some clever code

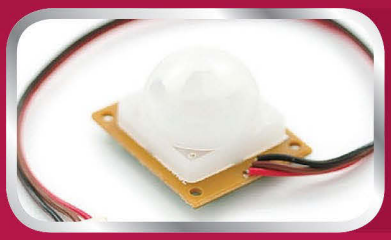
CONNECT PI TO A WATCH

Get alerts from your Raspberry Pi sent straight to your wrist, or smartphone, thanks to a little hacking with Pushbullet



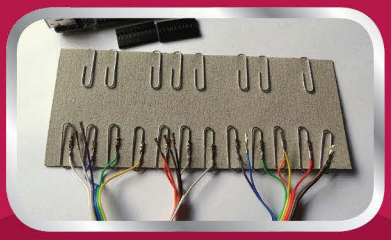
MINECRAFT TNT RUN

Create a Minecraft mini-game that tests your speed and reaction times as you outrun deadly virtual explosions



VIDEO TRIPWIRE

Troll your friends with a motion-sensing Rickroll, or just continue watching a playlist of interesting documen-nah... Rickroll



SPOON-O-PHONE

A custom Stylophone-style musical instrument made using a Raspberry Pi, some paper clips, and a teaspoon among other less amusing components

RICHARD HAYTER

Richard is a mentor at CoderDojo Ham, and his school Code Club was one of the winning teams in the Primary Astro Pi competition.
richardhayler.blogspot.co.uk / @rdhayler / coderdojoham.org

HIGH-SPEED PHOTOGRAPHY

WITH THE RASPBERRY PI CAMERA

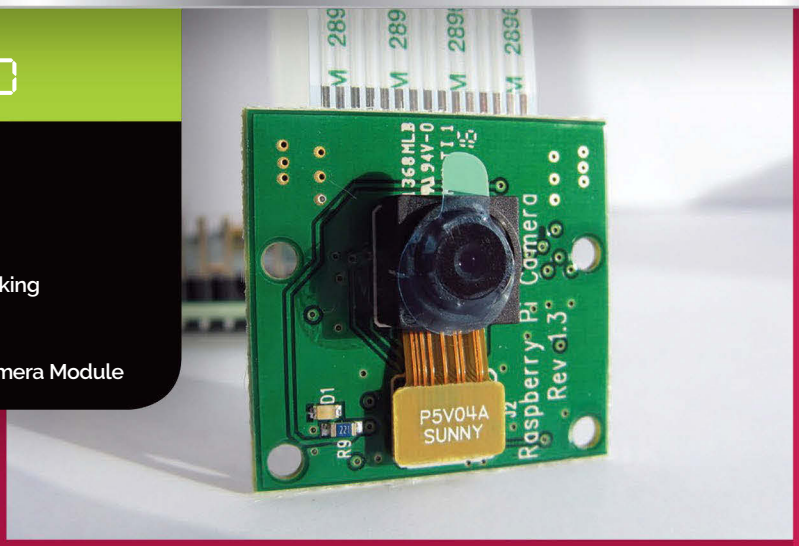
All you need to make dazzling slow-motion clips of exciting events is your Pi and the Camera Module!

INFO CARD

Difficulty
Easy

Type
Programming/Making

Requirements
Raspberry Pi & Camera Module



You've probably seen great clips on YouTube of water-filled balloons bursting in slow motion, and imagined you'd need really expensive camera equipment to capture anything similar. Not so! Your humble Raspberry Pi with a Camera Module is surprisingly capable when it comes to high-speed photography. Loads of everyday events look cool is slo-mo, but how about recording some dramatic crashes with toys?

At first glance it seems counter-intuitive, but in order to create a smooth slow-motion movie, you need a high-speed camera. Essentially, a movie is just a collection of still photos, or frames, all played one after the other at a speed that matches the original action. A slow-motion clip is produced by recording more frames than are normally needed and then playing them back at a reduced speed. Normal film is typically recorded at 24 frames per second (fps), with video frame rates varying between 25 and 29fps depending on which format/region is involved. So if you record at 50fps and play back at 25fps, the action will appear to be taking place at half the original speed. It's actually more complicated than that with the use of interlaced frames, but you don't really need to consider them here.

CLIPS CAN NOW BE RECORDED AT UP TO 90FPS

The original software for the Pi camera was limited in terms of the frame rates it could cope with, but a subsequent update in 2014 added new functionality so that clips can now be recorded at up to 90fps. There is one slight limitation: the high frame rates are achieved by combining pixels from the camera sensor, so you have to sacrifice resolution. So, although the Pi camera can record at a resolution of 2592×1944, a high-speed mode of 90fps is only possible at 640×480. This is still good enough to capture decent-quality images, and perfect for sharing online.

A quick way of getting started is to pick some everyday objects and record them in motion. How about a dropped egg hitting a table top? A pull-back toy car crashing through some Lego blocks? Or even the old favourite of a water balloon bursting (best to do this one outside)?

PICK SOME EVERYDAY OBJECTS AND RECORD THEM IN MOTION

Once you've chosen your subject, you'll need a way of holding and angling the camera, and some way of lighting the scene. Neither needs to be sophisticated: a normal desk lamp works fine for extra illumination indoors, while a 'helping-hand' work aid is brilliant for keeping the camera stable at tricky angles. You might also want to invest in a longer cable for the camera. You can get a 30cm ribbon cable for less than £2 or if you want to go even longer, a set of special adaptors allows you to extend using a standard HDMI cable.

A 'HELPING-HAND' WORK AID IS BRILLIANT FOR KEEPING THE CAMERA STABLE AT TRICKY ANGLES

The standard mode of operation for the Raspberry Pi camera is that a small red LED illuminates when recording is taking place. This can cause some undesirable reflections if the camera is positioned close to an object (e.g. a wall of Lego bricks). You can just block it off with a blob of Plasticine, or you can turn it off completely by adding the line **`disable_camera_led=1`** to your **/boot/config.txt** file.

The command for capturing video with the Raspberry Pi camera is **`raspivid`**, and this is best run from the terminal. There are a number of command options that you need to specify:

`-fps` sets the frame rate.

`-w` and **`-h`** specify the width and height of the frames. For the fastest frame rates, set this to 640 and 480 respectively.

`-t` allows you to set how long to record for. If you're working by yourself, the easiest way to avoid missing any of the action is to begin filming for a predefined period, giving yourself plenty of time to start things off manually.

`-o` specifies the file name to use for the saved movie.

`-n` disables preview mode.

So, putting that all together, the following commands would capture a 5-second clip at 60fps and save the resulting movie in the file **test.h264**:

```
raspivid -n -w 640 -h 480
-fps 60 -t 5000 -o test.h264
```

Right - now that you've recorded your movie clip, how can you play it back? One easy way is to use the free VLC Player, which you can install with:

```
sudo apt-get install vlc
```

The Pi version has some handy features which can be accessed by checking the 'Advanced Controls' option under the View menu. These include the extremely useful 'Frame by Frame' button . You can also alter the playback speed to slow things down even further.

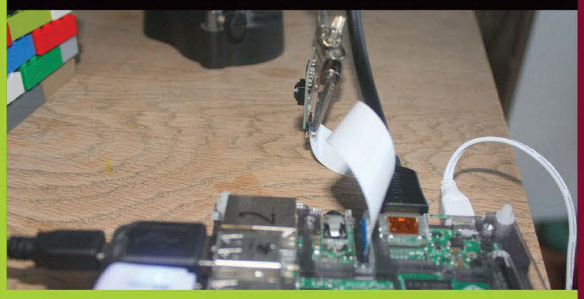
To extend the project, how about connecting a break-beam IR sensor pair via the GPIO pins and using these to trigger the camera recording? The Python Picamera library provides full access to the camera's functions and could be used with your code.

CAPTURING THE CLIP

STEP 1
Lights

Get your scene lined up and test how it looks using the camera preview mode for 5 seconds:

```
raspistill -w 640 -h 480 -t 5000
```

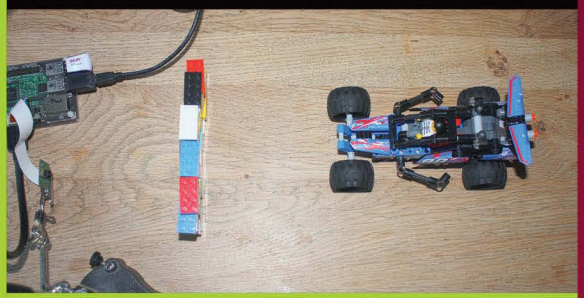


STEP 2
Camera

Type the command, ready for execution (but don't press **ENTER** yet):

```
raspivid -w 640 -h 480 -fps 90 -t 7000
-o myvid1.h264
```

Once triggered, this will capture a 7-second clip.



STEP 3
Action

When everything is ready, hit **ENTER** and then release the car/drop the egg/burst the balloon. You'll have footage before and after the event, which can be trimmed with some post-production editing.

LUCY HATTERSLEY

A technology writer and accomplished magazine editor, Lucy is well versed in all sorts of areas in the tech sector.
about.me/lucyhattersley
@lucyhattersley

CONNECT YOUR PI TO YOUR SMARTPHONE AND WATCH

Add Pushbullet to your Pi and get pings right on your wrist

INFO CARD

Difficulty
Medium

Type
Programming

Requirements
iPhone or Android phone (optional Apple Watch or Google Watch), Pushbullet account

One neat trick we recently discovered is how to hook up a Raspberry Pi to an Apple Watch. You do this using a service called Pushbullet, which normally syncs notifications from your desktop computer, but handily comes with an API for the Raspberry Pi. It's simple to set up and comes in useful for all kinds of projects.

Pushbullet is an app for the iPhone and Android that enables you to send notifications from your PC or Mac to your mobile device. It's mainly designed so people can sync alerts from their main computer to an iPhone or Android phone, and get notifications on their phone from their computer.

However, there is an API for Pushbullet that developers can use to include support for the service in their programs. This is great because it turns out you can easily create a script that sends you an alert with a custom message. And your Pi can easily call the script.

Why we really love this is because Pushbullet has support for the new smartwatches, such as the Apple Watch and Google Watch. With a bit of tinkering, you can get alerts sent from your Raspberry Pi directly to your wrist. How cool is that?

Hook up a Raspberry Pi to your Apple Watch to get alerts from your programs directly on your wrist

PROJECT OVERVIEW

You can set up Pushbullet on your phone and smartwatch, and use it to get your alerts from your Pi. The full Pushbullet API requires coding in Ruby, but we found it a lot easier to build a script in Unix that sends the alert, and then use the OS module in your Python programs to run the script.

The Pushbullet API works by using the **curl** command, along with your Access Token, to call the Pushbullet service. Your Access Token is a unique number that links the Pushbullet service with Pushbullet apps installed on your devices. You can learn more about the API at **docs.pushbullet.com**, but we'll walk you through it and show you how easy it is.

INSTALLING PUSHBULLET

Start by installing the free Pushbullet app on your Android phone or iPhone. When you first launch Pushbullet, you'll need to sign up with a Google or Facebook account – enter your email and password for that and tap Sign In. That's it for your phone and Apple Watch; everything else needs to be done on the Raspberry Pi.

THE ACCESS TOKEN

Hooking your Pi up to the app you've just installed requires an Access Token. This is a long alphanumerical key that is unique to your account.

pushalert.py and pushbullet.sh

Language
>UNIX SCRIPT
>PYTHON

DOWNLOAD:
github.com/lucy-hattersley/push-bullet-RPi.git

```
import os

buttonpush = raw_input("Press Enter to send alert...")
os.system('/usr/bin/pushbullet.sh "Alert from your Raspberry Pi"')
```

```
#/bin/bash

# replace [your access token] with the code from Pushbullet.com
# make sure this file is executable
# move file to /usr/bin

API="[Access Token]"
MSG="$1"
curl -u $API: https://api.pushbullet.com/v2/pushes -d type=note -d title="Raspberry Pi" -d body="$MSG"
```

You'll need it to ensure that your alerts are sent to the right person.

Open the Epiphany web browser and visit **pushbullet.com**. Log in using the same details as on your Android phone or iPhone. Now choose Settings>Account and scroll down to find the Access Token. You should test that this works before going any further (see 'Connecting Pushbullet' boxout).

The Raspberry Pi will connect to the Pushbullet API and send the alert. You should see a long message in the terminal, starting with **"active":true**, and receive a test notification on your smartphone. Open the Pushbullet app on your device to see the alert.

CREATING THE SCRIPT

If the test alert worked, you should create a script to send an alert whenever you want it. Your Access Token won't change, so you can create one script that accepts a custom alert as a command-line argument.

Enter **touch pushbullet.sh** and enter the code (above), replacing **[Access Token]** with your own code. You can also download it from GitHub (**bit.ly/1LKCgJ1**), but remember to edit it to insert your Access Token. The script needs to be executable, so enter **sudo chmod 755 pushbullet.sh** and move it to your **/usr/bin** folder (**sudo mv pushbullet.sh /usr/bin**).

Do a quick test to make sure everything worked. Enter **/usr/bin/pushbullet.sh "Testing Testing"** and you should get an alert saying 'Testing Testing'. All good? From now on, this script will ping you any message you include in the quotation marks.

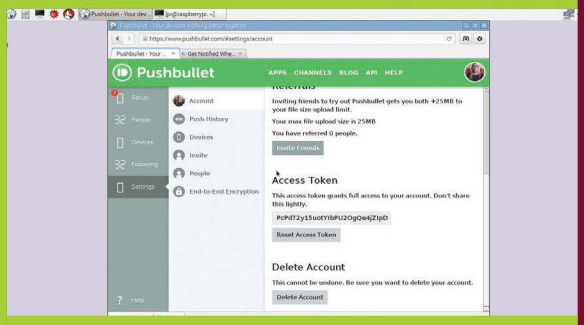
RUNNING THE SCRIPT FROM PYTHON

Now you have the script ready, you'll want to incorporate it into your programs. It's easy enough to do this in Python using the OS module. Enter **import OS** at the start of your program and **os.system ('/usr/bin/pushbullet.sh "[Your Alert]"')** to call the script from inside your program.

We've created a simple Python program to demonstrate how it works. Enter the code from **pushalert.py** and use **python pushalert.py** to run it. Now press **ENTER** and you'll get an alert sent to your iPhone or Android phone (and connected smartwatch).

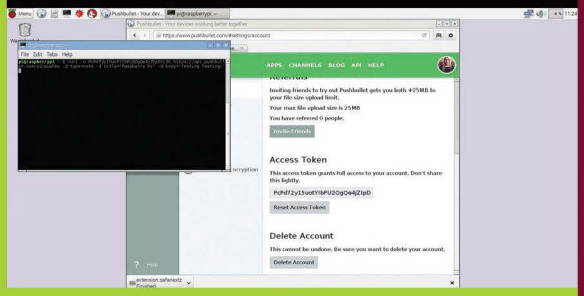
And that's how easy it is to get alerts on your wrist from your Raspberry Pi. Now you can include the OS call in any of your programs and get alerts whenever you want.

CONNECTING PUSHBULLET



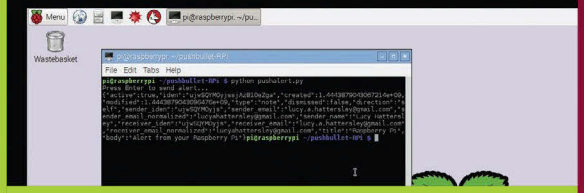
STEP 1
Getting your Access Token

First, install the Pushbullet app on your smartphone. Now, open the Epiphany web browser and visit **pushbullet.com**. Log in and choose Settings>Account, and scroll down to find your Access Token.



STEP 2
Test that it works

Test that your Access Token works correctly, using **curl -u [your access token] https://api.pushbullet.com/v2/pushes -d type=note -d title="Raspberry Pi" -d body='Testing Testing'.** Press **RETURN** and enter your Pushbullet password.



STEP 3
Action

Create a script called **pushbullet.sh** that makes the API call to Pushbullet (use the code we've provided with your Access Token). Use **sudo chmod 755 /usr/bin/pushbullet.sh** to make it executable and **sudo mv pushbullet.sh /usr/bin** to place it in your **bin** folder. Now you can call it from inside your Python programs.

JASPER HAYLER-GOODALL

The brains behind one of the winning Astro Pi entries, and a regular helper at CoderDojo Ham and Code Club. He's ten.
coderdojoham.org

TNT RUN!

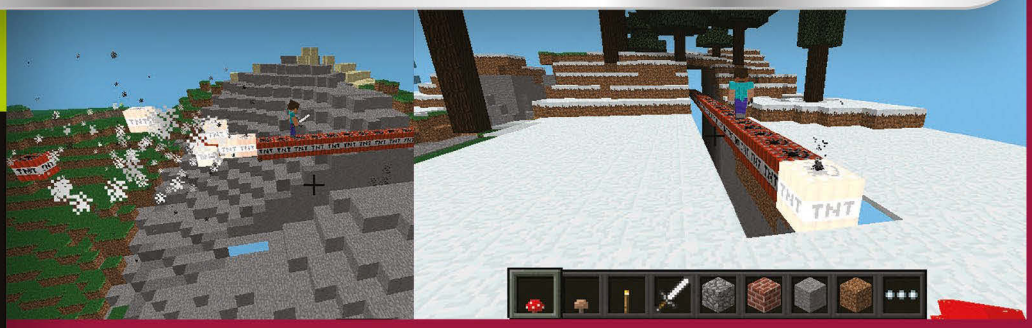
Can you outrun an explosion?
Test your speed with this marvellous Minecraft mini-game!

INFO CARD

Difficulty
Medium

Type
Programming

Requirements
Latest edition of Raspbian



Bash the first block to trigger the chain reaction along the TNT

One of the many amazing things about Raspberry Pis is that they have their own edition of *Minecraft* for free! And what's even better, you can code it in Python using the Minecraft API! In the next 30 minutes you will create a game called TNT RUN, in which you start at one end of a long line of TNT and have to make it to the safe area without the TNT exploding in your face.

The Minecraft API gives us complete control over many elements of the game. This includes teleporting players around the world and displaying helpful messages to the screen. We are also able to place blocks automatically – not just one at a time, but as a three-dimensional collection of blocks.

This game will also include a block which is unique to Pi and Pocket editions: invisible bedrock. We use this block to keep the TNT from falling to the ground when lit, and as an invisible path leading to the safe area. You may find something strange about this block when you place it next to a non-invisible block, so try experimenting with that - you need to look directly into the invisible bedrock.

TNT also behaves differently in the Pi edition. Whereas in other editions you set off TNT with flint and steel, a fire charge or a flaming arrow, in the Pi edition you just need to hit it a couple of times with anything. However, you can't just do this with any old TNT block: first, we need to set its block data value to an odd number (1, 3, 5, 7, or 9). Most blocks have a block data value and by changing this, it will alter the block's behaviour. For example, when you set Nether Reactor Core's block data value to 1, it appears in a red colour; if you set it to 2 then it will come out as a dark blue colour. We use these cool-looking blocks to

TNTRUN.py

Language
>PYTHON 2.7

DOWNLOAD:
github.com/snake48/TNTRUN

```
# import all the necessary modules
from mcpi.minecraft import Minecraft
from mcpi import block
import time

# connect with the Minecraft world
mc=Minecraft.create()

# get the player's position
pos=mc.player.getTilePos()

# check if the end of the world will engulf your
#creation & then move you if you're too close
if pos.z<-40:
    mc.postToChat(
    'teleporting to safer distance in progress!')
    mc.player.setPos(pos.x,pos.y,-40)
    pos=mc.player.getTilePos()

# mark where the teleport is
zpos=pos.z-40

# create the valley by hollowing it out with air
mc.setBlocks(pos.x-1,pos.y+3,pos.z,pos.x+1,pos.y-
7,pos.z-88,block.AIR.id)

#build the invisible bedrock support
mc.setBlocks(pos.x,pos.y-1,pos.z,pos.x,
pos.y-7,pos.z,block.BEDROCK_INVISIBLE.id)
mc.setBlocks(pos.x-1,pos.y-1,pos.z,pos.x,
pos.y-7,pos.z,block.BEDROCK_INVISIBLE.id)
mc.setBlocks(pos.x+1,pos.y-1,pos.z,pos.x,
pos.y-7,pos.z,block.BEDROCK_INVISIBLE.id)
mc.setBlocks(pos.x,pos.y-1,pos.z-88,pos.x-1,pos.y-
7,pos.z-88,block.BEDROCK_INVISIBLE.id)
mc.setBlocks(pos.x-1,pos.y-1,
pos.z-88,pos.x,pos.y-7,pos.z-88,
block.BEDROCK_INVISIBLE.id)
mc.setBlocks(pos.x+1,
pos.y-1,pos.z-88,pos.x,
pos.y-7,pos.z-88,block.BEDROCK_INVISIBLE.id)
mc.setBlocks(pos.x,pos.y,pos.z,pos.x,
pos.y-7,pos.z-92,block.BEDROCK_INVISIBLE.id)

# build the bomb
mc.setBlocks(pos.x,pos.y,pos.z,pos.x,pos.y,
pos.z-88,block.TNT.id,1)

# build the end podium
mc.setBlocks(pos.x-2,pos.y,pos.z-93,
pos.x+2,pos.y,pos.z-97,block.GLOWING_OBSIDIAN.id)
mc.setBlocks(pos.x-1,pos.y+1,pos.z-94,pos.x+1,
pos.y+1,pos.z-96,block.NETHER_REACTOR_CORE.id,1)
mc.setBlock(pos.x,pos.y+2,pos.z-95,
block.REDSTONE_ORE.id)

# set how many teleports you have
teleport=1

# build the display teleport signal block
mc.setBlock(pos.x+1,pos.y+1,pos.z-44,
block.NETHER_REACTOR_CORE.id,2)
mc.setBlock(pos.x-1,pos.y+1,pos.z-44,
block.NETHER_REACTOR_CORE.id,2)


# teleport player when at a certain position
while teleport ==1:
    pos=mc.player.getTilePos()
    if pos.z==zpos:
        mc.player.setPos(pos.x,pos.y,pos.z-24)
        teleport=0
```

HOW TO PLAY:

- The start of the TNT line will appear directly underneath you.
- The TNT won't just light itself; you need to detonate the first block of TNT by hitting it a couple of times.
- Once you've lit it (you'll know when it starts flashing white), turn around and then run for your life!
- Try to get to the teleporter, as it will give you an 18-block boost.
- If the TNT catches up with you, you will get flung up into the air and lose.

mark where the teleporter is and as a part of the end podium. The teleporter function in the code is designed so that if you manage to get to a certain point along the line of TNT, it gives you a boost and teleports you forward.

When you create your *Minecraft* world, fly around it and find a cool location to create your TNT course (i.e. not when you are near a cliff or the end of the world). Your code takes the player's position and constructs the TNT course, using this location as the starting point. When you run the code, make sure *Minecraft* is running, otherwise your code will give you a connection error.

Once you have finished the project, you can customise your own version of the TNT Run game by adding changes like making the row of TNT longer, or creating a fancier safe area. Also, when you're connected to a network with other Raspberry Pis, you can join someone else's world, create two lines of TNT and race to the safe areas simultaneously!

This is the safe area you need to run to. If you make it there, you've won

The dark blue Nether Reactor Core marks the location of the teleporter

LUCY HATTERSLEY

A technology writer and accomplished magazine editor, Lucy is well versed in all sorts of areas in the tech sector.
about.me/lucyhattersley
@lucyhattersley

INFRARED VIDEO ALARM

Use a PIR motion sensor to play a video clip when it detects an intruder

INFO CARD

Difficulty
Easy

Type
Making

Requirements
PIR motion sensor, breadboard, 10kΩ resistor, MP4 video clip

A PIR motion sensor detects movement and can be used to trigger an action on your Raspberry Pi

PIR motion sensors are a common type of security device. Standing for 'passive infrared', these sensors normally lurk in the corner of a room and sometimes light up as you walk past. During the day they don't tend to do anything, but sneak around at night and they may well set off the security alarm. In this project we'll be hooking a PIR sensor up to the Raspberry Pi to trigger a Rickroll whenever someone walks past!

USING SENSORS

Smaller PIR motion sensors are an easily available component that can be attached to the Pi. In this project we used one (Product Code: 312) from Cool Components (**coolcomponents.co.uk**).

PIR sensors come with three wires. You attach one to the power, one to ground, and one to a GPIO pin. When you power up the PIR sensor, it takes a couple of seconds to get a snapshot of the default levels in the room. Then, if anything changes the infrared levels because a person has moved in front of the sensor, the alarm pin will go low.

Which cable is power, ground and alarm will vary on each device. On our PIR unit, if you look at the dome with the wires pointing towards you, the middle wire is ground, the right wire is alarm, and the left is live. The colours can vary but typically they will be red (live), black (ground), and brown (alarm). Check the instructions to find out which layout your model is using.

Speaking of the dome, that's a Fresnel lens. This is used to magnify the infrared light and make the PIR sensor more effective. The actual infrared sensor is beneath it.

CONNECTING THE PIR SENSOR

You could hook up the PIR sensor directly to your Raspberry Pi, but it's a good idea to connect it via a breadboard. Connect the live to the positive rail and the ground to the negative rail, and place the alarm cable into any spare pin.

Now connect the breadboard to the Raspberry Pi. Use a black jumper wire to connect GPIO GND (pin 6) to the negative rail, and a red jumper wire to connect GPIO 5V (pin 2) to the positive rail. Connect GPIO 17 (pin 11) to the same rail as the alarm cable.

So that's our trigger set up: the PIR sensor will power up, set its

piralarm.py

Language
>PYTHON

DOWNLOAD:
github.com/lucyhattersley/piralarm.git

```
# detects motion sensor from PIR Sensor Module attached to GPIO 17.
# change [MediaFile] in path to name of media file in Music folder.

import os
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)

GPIO.setup(17, GPIO.IN)

while True:
    if GPIO.input(17):
        os.system("omxplayer -o hdmi /home/pi/Music/[Media File].m4v")
```

levels to the current room, and monitor for any change. When it detects one, it will drop the input on GPIO 17 to low.

What we need now is a means to play our video clip.

GETTING A VIDEO ROLL

You can play any video clip you want, or even use the Camera Module to record your own message, but we decided to go with Rick Astley's *Never Gonna Give You Up* and build an automatic Rickroll alarm. We're using a video in MP4 format that we bought from the iTunes Store; this is DRM-free and we transferred it to the Music folder on our Raspberry Pi using a USB thumb drive.

We're going to play this file using omxplayer. You should have omxplayer installed as default in Raspbian Jessie – enter **`omxplayer --version`** to check it's installed. If not, enter **`sudo apt-get update`** and then **`sudo apt-get install omxplayer`**.

Move into the folder with the video clip; we've placed ours in the Music folder as it's a music video, but you can put yours into any folder. Now enter **`omxplayer -o hdmi [filename]`** to play the video clip. The **`-o hdmi`** option ensures that the audio is outputted through the HDMI output; it may work without this, but it's best to include it just in case.

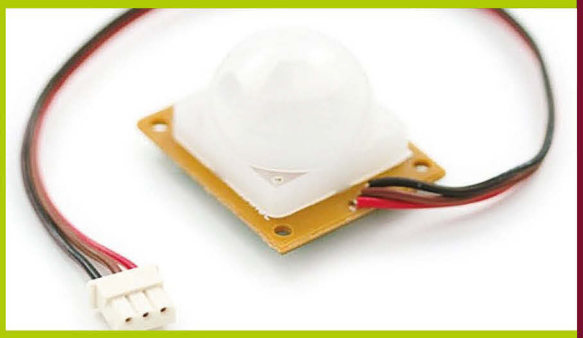
The video should start playing in a full-screen window on top of your desktop, or full-screen if you are in the command line. Omxplayer is a handy tool to have, and you can use it as a regular media player. Press **P** or **SPACE** to pause the video (and again to resume it), and use the **+** and **-** keys to increase or decrease the volume so your alarm is at the level you want. Press **ESC** when you've had enough of the video. Incidentally, you can use **`omxplayer -k`** to view all of the shortcut keys.

Now use **`mkdir`** to create a folder for the code, and use **`touch`** to create a file called **piralarm.py**. Use a text editor to add the code supplied, or clone it using the GitHub link provided. Make sure you change the **`[Media File]`** part to the name of your video clip.

Enter **`python piralarm.py`** to arm and set your alarm. Now, when a person moves, they will trigger the code and play your video.

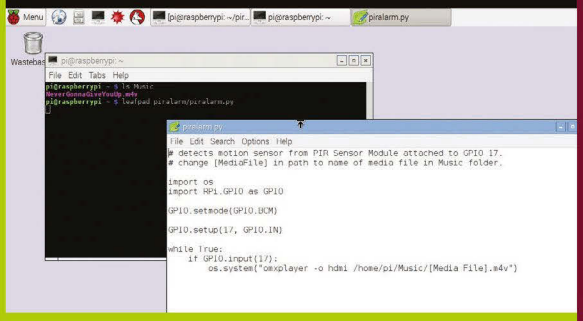
You'll probably find the PIR sensor too effective at first. Some PIR motion sensors come with a potentiometer that can be used to adjust the sensitivity of the device. One it's set to the correct level, you can leave it to trigger whenever anybody walks into your room.

BUILDING A RICKROLL ALARM



STEP 1
Connect the PIR

First, you need to connect a PIR sensor to your Raspberry Pi. There are three wires. Connect live to 5V, neutral to GND, and alarm to GPIO 17.



STEP 2
Get the code

Transfer a video clip to the Raspberry Pi and create a file containing the Python code. Make sure the **piralarm.py** code links to the full path of your video clip.

STEP 3
Video (Rick)roll

Once the code is running, the PIR motion sensor detects any movement, and then triggers the video to play. We're using the Rickroll video as a demo, but you can use any video clip you want.

MIKE COOK

Veteran magazine author from the old days and writer of the Body Build series. Co-author of *Raspberry Pi for Dummies*, *Raspberry Pi Projects*, and *Raspberry Pi Projects for Dummies*.
bit.ly/1aQqu15

THE SPOON-O-PHONE

Make your own version of the famous Stylophone, but this time play it with a teaspoon

INFO CARD

Difficulty
Medium

Type
Making

Requirements

- B+ or Model 2 Raspberry Pi
- 20× large plain metal paper clips (32mm long)
- Teaspoon
- M3 nut and bolt
- 22× jumpers with crimped female headers
- 4× 10-way header shell
- 2× pieces of cardboard 170×70mm

The Spoon-o-phone connected to your Raspberry Pi. Play the notes by touching the paper clips with the teaspoon

Create your own version of the classic Stylophone, without the need for any electronic components or soldering.

HOW IT WORKS

The paper clips are wired directly to some GPIO pins, and the spoon to a ground pin. The software sets the GPIO pins to be inputs, with the internal pull-up resistors enabled; they will normally read back as a logic 1. However, if the grounded spoon touches a paper clip, that input will now read as a logic low. When the code sees this, it will trigger the playing of a sample for that note. Like the original Stylophone, the Spoon-o-phone has a vibrato effect, turned on and off with a mouse click in the small on-screen window. With the vibrato effect on, a different set of samples are selected. You need a Model 2 or B+ Pi for this project as it needs 20 input pins, but you could make it with older Pi models if you cut down the range or omitted black notes.

MATERIALS USED

The 'Make your own Cable Kit' (**bit.ly/1NTIGb7**) was used for the crimped wires and shells, but any other method will do just fine. The cardboard back of an A4 writing pad is ideal; it has to be thin enough to take the paper clips. Make sure the holes on the shells are on the outside so you can change the wires over if you make a mistake. Fix the ground wire to the teaspoon with an M3 nut and bolt.

Once the keyboard is finished and tested, tuck all the wires under the cardboard and make a sandwich by hot-melt-gluing another piece of cardboard to the underside.

SOFTWARE

Get the sound samples from **bit.ly/1LnvTtK** or the GitHub. The code is written in Pygame, but uses a music system designed for continuous background audio: as the Stylophone samples are very short, they must be looped so a note can be played for as long as you hold the spoon on a clip.

The GPIO pins used are in the list `pinList` and can be shuffled about if you get your wiring wrong. The `notes` list contains the names of the sample files for each note, with a 'v' appended for vibrato. They all sit in a **stylo_smpl** directory at the same level as the main program. Small `time.sleep` delays are used to minimise any contact bounce effects on the input pins.

spoon_o_play.py

Language
>PYTHON 2.7

DOWNLOAD:
github.com/Grumpy-Mike/Mikes-Pi-Bakery/tree/master

```
#!/usr/bin/python
# Spoon-o-phone - Stylophone emulator By Mike Cook - Sept 2015
import pygame, time, os
import wiringpi2 as io

pygame.init()
os.environ['SDL_VIDEO_WINDOW_POS'] = 'center'
pygame.display.set_caption("Stylophone")
screen = pygame.display.set_mode([190,40],0,32)
pygame.mixer.quit()
pygame.mixer.init(
frequency=22050, size=-16, channels=2, buffer=512)
pygame.event.set_allowed(None)
pygame.event.set_allowed(
[pygame.MOUSEBUTTONDOWN,pygame.KEYDOWN,pygame.QUIT])
font = pygame.font.Font(None, 36) ; vibrato = False
pinList = [21,26,20,19,16,13,6,12,5,11,7,8,9,25,10,24,
22,23,27,18] # GPIO pins
notes = ["a2","a#2","b2","c3","c#3","d3","d#3","e3","f3",
"f#3","g3","g#3","a3","a#3","b3","c4","c#4","d4","d#4","e4"]

def main():
   initGPIO()
   print"Stylophone - By Mike Cook"
   drawWords("vibrato off",36,6)
   while True:
      checkForEvent()
      for pin in range (0,len(pinList)):
        if io.digitalRead(pinList[pin]) == 0:
           sound = getSample(pin)
           pygame.mixer.music.load("stylo_smpl/"+sound+".wav")
           pygame.mixer.music.set_volume(1.0)
           pygame.mixer.music.play(-1,0.0)
           time.sleep(0.030)
           while io.digitalRead(pinList[pin]) == 0:
               pass
           pygame.mixer.music.fadeout(100)
           time.sleep(0.030) # debounce time

def getSample(number):
   sample = notes[number]
   if vibrato :
      sample +="v"
   return sample

def initGPIO():
   try :
      io.wiringPiSetupGpio()
   except :
      print"start IDLE with 'gksudo idle'
      from command line"
      os._exit(1)
   for pin in range (0,len(pinList)):
      io.pinMode(pinList[pin],0) # make pin into an input
      io.pullUpDnControl(pinList[pin],2) # enable pull-up

def drawWords(words,x,y) :
        textSurface = pygame.Surface((len(words)*12,36))
        textRect = textSurface.get_rect()
        textRect.left = x ; textRect.top = y
        pygame.draw.rect(screen,(0,0,0),
        (x,y,len(words)*12,26), 0)
        textSurface = font.render(
words, True, (255,255,255), (0,0,0))
        screen.blit(textSurface, textRect)
        pygame.display.update()

def terminate(): # close down the program
    pygame.mixer.quit() ; pygame.quit()
    os._exit(1)

def checkForEvent(): # keyboard commands
    global vibrato
    event = pygame.event.poll()
    if event.type == pygame.MOUSEBUTTONDOWN:
       vibrato = not vibrato
       if vibrato:
          drawWords("vibrato on ",36,6)
       else:
          drawWords("vibrato off",36,6)
    if event.type == pygame.QUIT :
        terminate()
    if event.type == pygame.KEYDOWN :
       if event.key == pygame.K_ESCAPE :
          terminate()

# Main program logic:
if __name__ == '__main__':
    main()
```

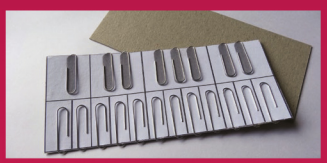
BUILDING THE SPOON-O-PHONE

STEP 1
The keyboard layout

Mark out a piece of paper with the lines separating each key, and also shading for the black notes. You can draw this or, if you like, print out the keyboard PDF file found in our GitHub repository. Stick this onto a piece of cardboard with white stick glue.

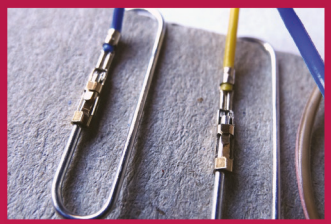
STEP 2
Fitting the paper clips

Once the layout is ready, fit the paper clips. Make sure they are all the same way round, with the long side to the front of the card.



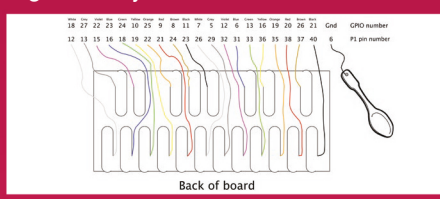
STEP 3
Wire up the paper clips

Turn the board over and push the crimped connectors into the end of the paper clips. Fix them with hot-melt glue after testing the finished board.



STEP 4
Wire up to the Pi

Make the connections between the paper clips and the GPIO header shells. Push them into an unpowered Pi and hot-melt glue all the shells together so they align correctly.



SUBSCRIBE TODAY!

Subscribe to the Official Raspberry Pi mag today for a whole host of benefits

Subscription benefits

- Save up to 25% on the price
- Free delivery to your door
- Never miss a single issue

NEW US SUBS OFFER!
imsnews.com/magpi

SAVE UP TO **25%**

Pricing

Get six issues:

£30 (UK)

£45 (EU)

$69 (US)

£50 (Rest of World)

Subscribe for a year:

£55 (UK)

£80 (EU)

$129 (US)

£90 (Rest of World)

Direct Debit

£12.99 (UK) (quarterly)

How to subscribe:

- bit.ly/MagPiSubs (UK - ROW)
- imsnews.com/magpi (US)
- Call +44 (0)1202 586848
- Use the form to the right

Available on the App Store

Get it on Google play

SUBSCRIPTION FORM

YES! I'd like to subscribe to The MagPi magazine & save money

This subscription is: ☐ For me ☐ A gift for someone*

Mag#39

YOUR DETAILS Mr ☐ Mrs ☐ Miss ☐ Ms ☐

First name ..... Surname .....

Address .....

.....

Postcode ..... Email .....

Daytime phone ..... Mobile .....

**If giving* The MagPi *as a gift, please complete both your own details (above) and the recipient's (below).*

GIFT RECIPIENT'S DETAILS ONLY Mr ☐ Mrs ☐ Miss ☐ Ms ☐

First name ..... Surname .....

Address .....

Postcode ..... Email .....

PAYMENT OPTIONS

1 DIRECT DEBIT PAYMENT £12.99 every 3 issues (UK only)
Instruction to your bank or building society to pay by Direct Debit

DIRECT Debit

Please fill in the form and send to:
The MagPi, Select Publisher Services Ltd,
PO Box 6337, Bournemouth BH1 9EH

Service user number 8 3 8 7 7 3

Name and full postal address of your bank or building society:

To: The Manager Bank/building society .....

Address .....

.....

..... Postcode .....

Name(s) of account holder(s) .....

Branch sort code ☐☐☐☐☐☐ Account number ☐☐☐☐☐☐☐☐

Reference ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐ (Official use only)

Instruction to your bank or building society
Please pay Select Publisher Services Ltd Direct Debits from the account detailed in this instruction subject to the safeguards assured by the Direct Debit Guarantee. I understand that this instruction may remain with Select Publisher Services Ltd and, if so, details will be passed electronically to my bank/building society.

Signature ..... Date ☐☐/☐☐/☐☐

Banks and building societies may not accept Direct Debit instructions for some types of account.

SUBSCRIPTION PRICING WHEN PAYING BY CHEQUE OR CREDIT/DEBIT CARD

6 ISSUES ☐ UK £30 ☐ Europe £45 ☐ Rest of world £50

12 ISSUES ☐ UK £55 ☐ Europe £80 ☐ Rest of world £90

2 CHEQUE

I enclose a cheque for ..... (made payable to Select Publisher Services Ltd)

3 CREDIT/DEBIT CARD ☐ Visa ☐ MasterCard ☐ Maestro ☐ Switch

Card number ☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐☐

Expiry date ☐☐ ☐☐ Valid from ☐☐ ☐☐ (if shown)

Issue number ☐☐ (if shown) Security number ☐☐☐
(last 3 digits on the back of the card)

Signature ..... Date ☐☐/☐☐/☐☐

I would like my subscription to begin from issue ..... (month + year)

RETURN THIS FORM TO:
MagPi Magazine Subscriptions, Select Publisher Services Ltd, PO Box 6337, Bournemouth BH1 9EH

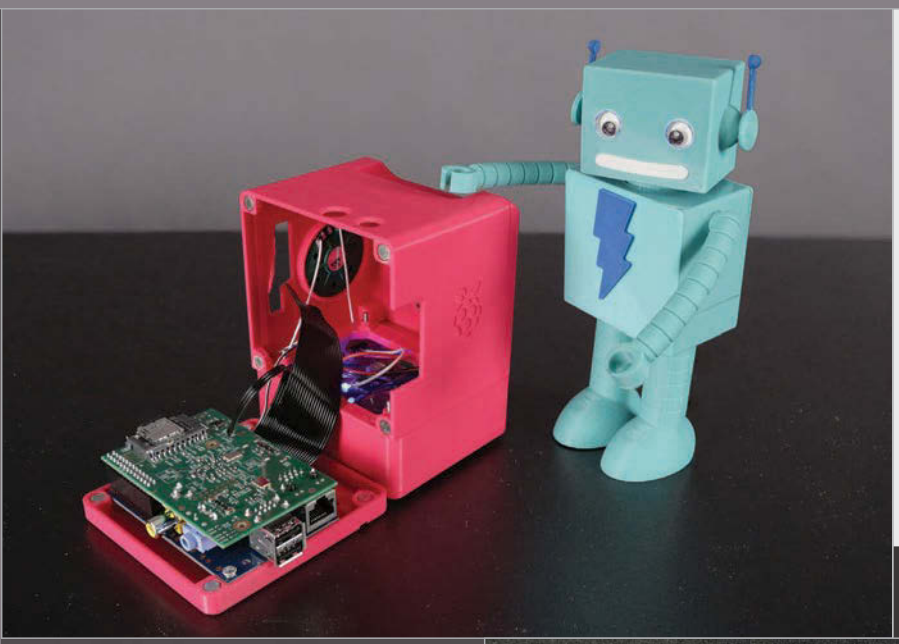
☐ Please tick this box if you DO NOT want to receive any other information from Select Publisher Services Ltd.

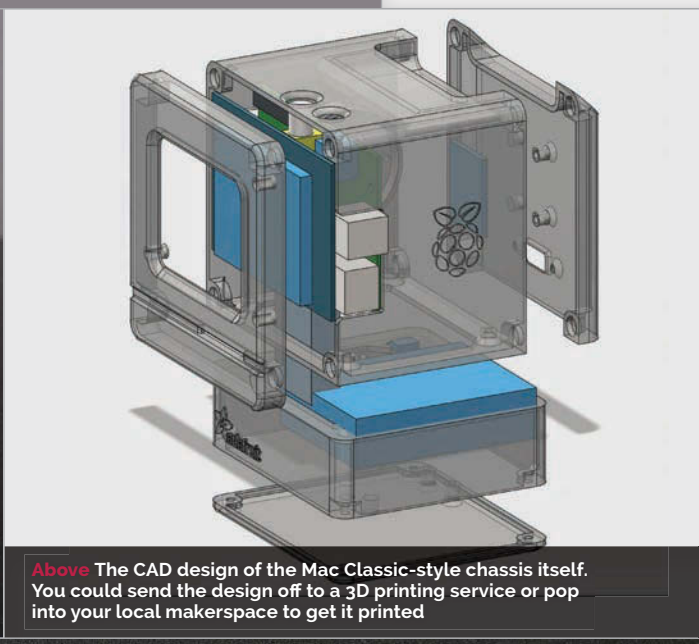
☐ Please tick this box if you DO NOT want to receive any other information from other companies.

☐ Please tick this box if you DO NOT want to subscribe to The MagPi newsletter.

MAC MINI PI

Arguably the smallest, cutest classic Mac-inspired project, the Mac Mini Pi is actually much more than a swanky case for your Raspberry Pi...



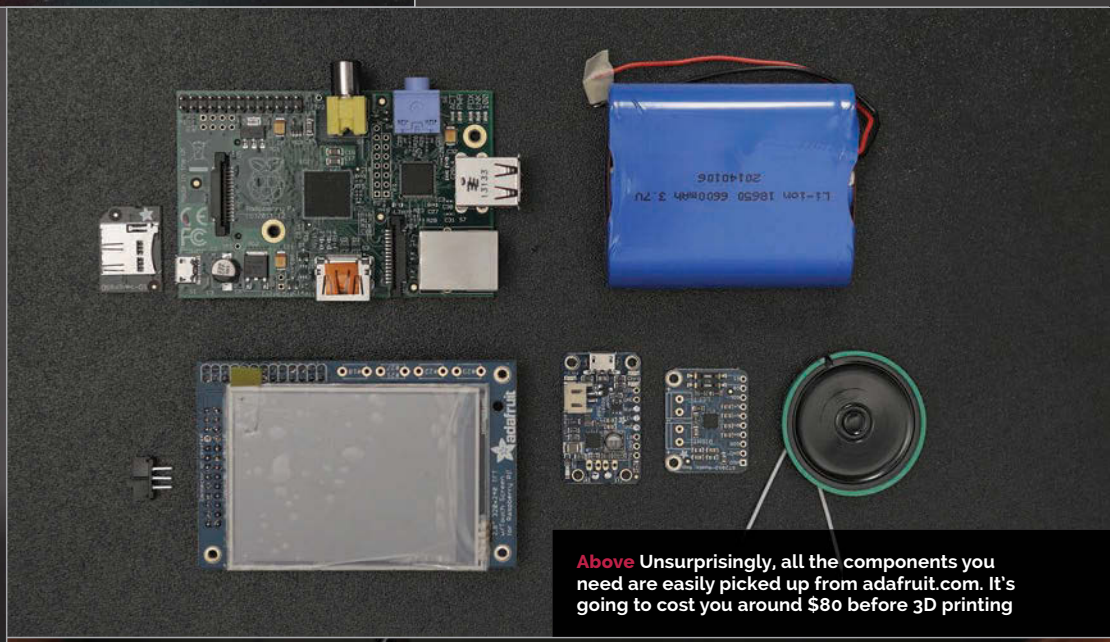


Above The CAD design of the Mac Classic-style chassis itself. You could send the design off to a 3D printing service or pop into your local makerspace to get it printed

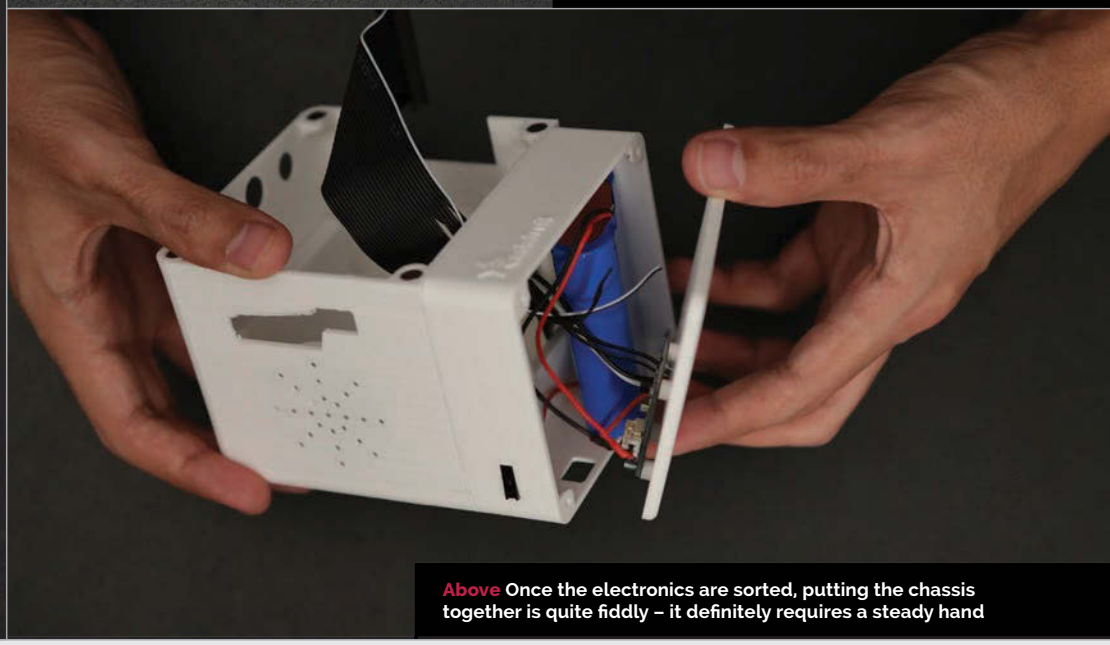
Adafruit knows a thing or two about building impressive Raspberry Pi projects, and the Ruiz brothers' Mac Mini Pi – originally inspired by RetroMacCast's John Badger – is certainly no exception. Besides being an unashamedly cute design, this tiny Mac is much more than a fancy case for your Raspberry Pi – it's a fully functioning emulator, so you can relive those Mac OS 7 glory days (or experience them for the first time). The software used for emulation is a flavour of Mini vMac (**minivmac.sourceforge.net**) that perfectly emulates early Macs that ran Motorola's 680x0 microprocessor. The software runs happily alongside Raspbian, too.

Adafruit.com's guide includes everything you need to build the project yourself, including detailed instructions and the files you need for the 3D print (you can use a 3D printing service if you lack a printer of your own). Under the hood, the project uses a standard Raspberry Pi Model B, the 320×240 3.2″ PiTFT screen, a 3.7V lithium-ion battery, and a simple audio amp, among other things. Some basic soldering skills are a must and you can expect to spend most of the weekend on the build itself (it's fiddly!).

You can check out this amazing project in its entirety at: **learn.adafruit.com/mini-mac-pi**



Above Unsurprisingly, all the components you need are easily picked up from adafruit.com. It's going to cost you around $80 before 3D printing



Above Once the electronics are sorted, putting the chassis together is quite fiddly – it definitely requires a steady hand

MATT REED

Matt Reed lives in Nashville, Tennessee, and is a creative technologist at RedPepper. **mcreed.com**

MASON JAR PRESERVE

Quick Facts

- The Mason jar was invented in 1858 by tinsmith John L Mason
- You can buy Mason Jars in the UK from Kilner – **kilnerjar.co.uk**
- Don't overclock the Raspberry Pi, as it could overheat
- The only storage limitation is what you attach to the Raspberry Pi
- Matt used Node.js to get the LEDs to work in time with BitTorrent

Matt Reed backs up his family photos in a Mason jar. He tells Lucy Hattersley how to preserve digital memories in style

Matt Reed, a professional web developer and maker from West Virginia, has created this amazing project. The Raspberry Pi-powered Mason Jar Preserve is the most stylish backup solution we've ever seen.

At this point, *The MagPi*'s non-US readers might be wondering what on earth a Mason jar is. "They are industrial-grade glass jars with a sealable lid that were originally used at home to preserve foods throughout the winter," says Matt. "Mason jars allowed for foods that were harvested in the summer to last all year round. Who doesn't want tasty fried okra in February?

"In my great-grandmother's basement, there were shelves and shelves of various-sized jars filled with everything from pickled beets to you-name-it preserves." But rather than preserve tasty foodstuffs, Matt's Mason jar preserves Matt's family memories.

Canning a Raspberry Pi

The idea came to Matt at work: "I had a few various-sized Mason jars sitting around. One day I just grabbed the largest jar and slid the Pi inside. It fit, even with a little room to spare!" Building the Mason Jar Preserve was fairly

PRESERVING YOUR PI

>STEP-01
Building the base
The Raspberry Pi fits inside the Mason jar, but it needs to be held upright. For this, Matt cut a six-inch base. He then sanded it so it had round edges (reminiscent of an Apple TV).

>STEP-02
Put a lid on it
The lid of the Mason jar has a hole drilled in the centre. This is used to secure it with a nut and washer. Then four more holes are drilled to house the LED lights. The power and Ethernet cables are run through another hole.

>STEP-03
Putting it together
The BitTorrent software is installed and set up to back up a folder on other local computers. Finally, the Mason jar is screwed on top of the lid and is lit up by the LEDs.

straightforward: "I used a saw to cut the base into a square, a sander to round it off, and a drill to make the LED, Ethernet, and power holes. A few trips to Home Depot for antique drawer pulls, some rubber feet, brackets and screws, and that was it."

Matt's Mason Jar Preserve connects via Ethernet, but you could use WiFi. "I am still curious how much glass would affect the signal," he tells us. "In the end I chose Ethernet for transfer speed, availability, simplicity, and reliability. Also, because I was low on USB WiFi dongles. Like, zero."

Powered by BitTorrent

Matt used BitTorrent to create the backup software. "It's similar to Dropbox," he says, "but instead of a centralised server in the cloud, you connect two or more of your own devices directly together over the BitTorrent protocol. Just like Dropbox, connected BitTorrent sync machines all sync up a specified folder on their system. So basically, it's free and the storage is only limited to the connected storage."

Unlike BitTorrent file sharing, Matt's system is completely private. "Whenever you create a sync folder, it gets a unique string of letters and numbers," explains Matt. "This key uniquely identifies it over the BitTorrent sync protocol. Keys can provide either read/write access or read-only access to anyone you give it to. You generally want to keep them safe. If I shared that key publicly, then I'd have a public file-sharing seed similar to the BitTorrent everyone knows and loves."

Matt uses his the Mason Jar Preserve to back up family photos and videos. "I really just wanted one more layer of redundant backup of things like the birth of our kids, wedding photos, and various family-related media. It uses the Raspberry Pi's SD card for storage. Therefore, the larger the SD card you put in your Pi, the more you can store."

We think the end result is glorious. When the Mason Jar Preserve is syncing, it animates the LEDs using the Node.js script, BitTorrent Sync API, and the GPIO pins. When it sits idle, they glow red. "I wanted it to look like the jar was filled with red raspberry preserves," says Matt. "It especially looks great at night."

Some furniture fittings from Home Depot were used to embellish the design

The Mason jar looks especially good in low light when its LEDs are illuminated

YVES DAIGLE

An aerospace engineering officer for the Royal Canadian Air Force whose job is currently to design and build flight simulators.
github.com/whyvas/raftBerry

For a watercraft, this is quite a simple selection of controls to get it work, all hooked to the Pi

Cruise in style, with music and DVDs to keep you entertained out on the water

The most important part about such a craft is that it floats. And it does

RAFTBERRY

Escape a desert island with your volleyball BFF in style, or just pootle around a lake with some pals, with this Raspberry Pi-powered raft

Quick Facts

- The raftBerry had been planned for about two years
- Yves hasn't programmed since university
- The raftBerry has all the required boating safety equipment
- Yves wants to add crazy lighting and 'look to steer' systems
- For the military, Yves has used 22 Raspberry Pis

Picnics are fun. You fish a cooler bag from the back of a cupboard along with some half-forgotten freezer blocks from the ice tray and fit in as many sandwiches, sausage rolls, and cans of soda as you can. After grabbing some family or friends, you find a nice spot and eat outside. Wonderful, aside from the inevitable invasion by ants, but still great. What if you could take ants out of the equation, though? You could if you were somehow able to have a picnic out in the middle of a lake. Meet Yves Daigle and his raftBerry project...

"The idea is to create a fully autonomous electrically propelled floating dock, controlled by a Raspberry Pi, to drive us around our lake," Yves tells us. As you can see from the resulting picture of Yves and some friends enjoying a meal on the floating dock, the idea has come to fruition.

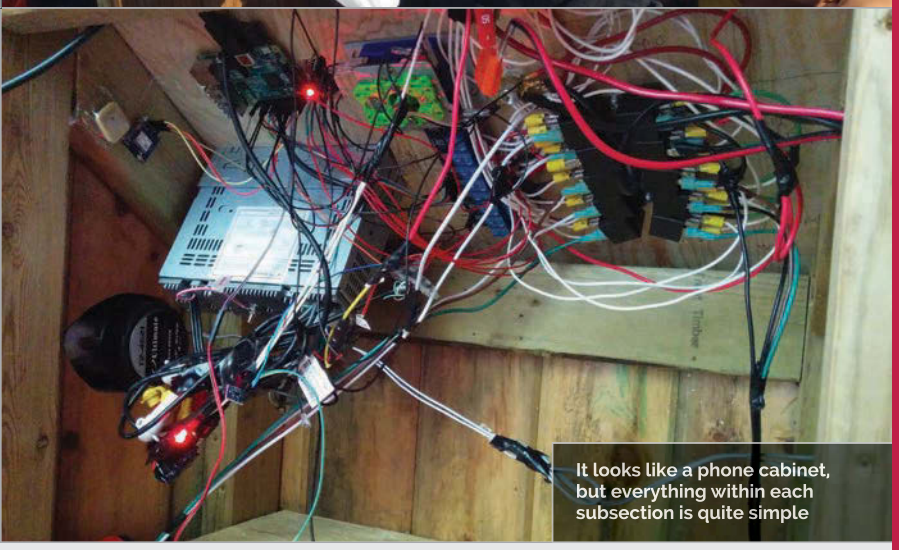
As an aerospace engineer serving in the Canadian Air Force, Yves is used to creating complex equipment, so it's no surprise that this side-project is not exactly simple as well:

"The control system has two available modes that can be selected via a switch on the console: manual or autonomous. The dock uses two static motors for propulsion that can also steer the dock via differential thrust.

"The Pi reads inputs via the GPIO from the arcade joystick, emergency shutdown button, and mode select switch. Each motor is independently controlled by five automotive relays, three of which are used to set the speed, and the remaining two are used in an H-bridge configuration to set the motors in forward or reverse. The GPIO output pins control a small eight-channel relay board, which in turn controls these ten automotive relays. The whole system, from motors to control panel, is designed to be removed from the dock after every trip, using a hand truck."

The whole thing took him a month to finish after he started to properly work on it and also includes a DVD player, a table, seating, and a camping stove. Less of a picnic, then, and more a floating campsite.

A floating picnic for everyone, thanks to a little hacking



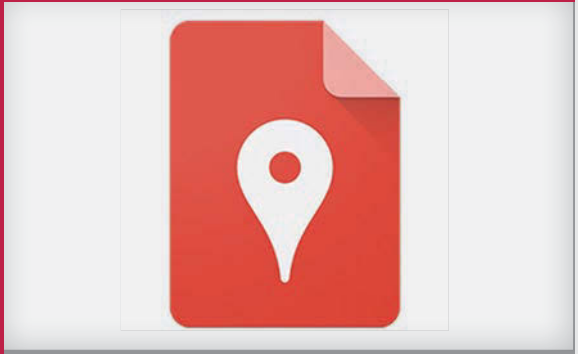
It looks like a phone cabinet, but everything within each subsection is quite simple

"Manual mode works flawlessly," Yves explains. "The relays do get quite hot and full speed appears to not be as powerful as it was with the original control head. I suspect the relays are nearing max load and are causing some voltage drops. I plan to acquire larger relays. The motors may be a little underpowered."

Automatic mode doesn't quite work right now, but Yves is working on it. As well as using Raspberry Pis at home and on watercraft, he's been using them in his job: "We have 22 of them performing various tasks at the moment. I've found that Pis are several orders of magnitude more cost-effective than commercial alternatives. Saving taxpayer dollars is something I take quite seriously."

If you want to see some of the build, along with the raftBerry in action, you can check out the video of it on YouTube: **youtu.be/FZvU3U1wZWo**

AUTOMATICALLY DRIVING MR DAIGLE



>STEP-01

Plan a route

Auto mode should work by allowing you to first create a custom map in Google Maps, then exporting the map to a .kml file the raftBerry can use.

>STEP-02

Start to navigate

The raftBerry uses GPS data to compare your location to the first waypoint, and produces a method to move to the waypoint. This is updated once per second.

>STEP-03

Enjoy the ride

The raftBerry waits until you're within ten metres of the waypoint before calculating the route to the next one. Keep an eye out for anything in your path, though!

CHRIS DUERDEN

A mechanical engineer with a fresh law degree, and experience of a lot of electronic projects and systems.
imgur.com/a/8u06E

If you're sick of bad *Super Mario Bros 3* levels in *Super Mario Maker*, try out the original on this 3.5″ screen with many other games

There are enough buttons to play all major home consoles up to the PlayStation

A mixture of old and new: a GBA volume control and USB input for loading ROMs

SUPER GAMEGIRL

Quick Facts

- This was Chris's first Raspberry Pi project
- The volume wheel is from a Game Boy Advance
- Chris created custom PCBs for this project
- Unfortunately, it doesn't run Nintendo 64 games well
- Chris's favourite game to play on it is *Super Mario Bros 3*

Want a better handheld games console? The answer may be to make one yourself with a Raspberry Pi and a 3D printer

Over the past few issues, we've covered a handful of retro gaming projects of varying types: a refurbished SNES, a 3D-printed NES, a Power Glove running games, and so on. While these are all definitely cool projects, you know what's *really* cool? A portable retro gaming machine! In the past, this spot has typically been taken up by the very hackable Sony PSP, but Chris Duerden made the decision to go down a much more homebrew route by making his machine from scratch using the same Raspberry Pi that powers those other retro gaming projects.

"My project is a Raspberry Pi 2-powered Game Boy-looking thing," Chris tells us. "It runs RetroPie, so it can play games from a lot of different systems. The goal was to have enough buttons to support PlayStation games, have a 3.5″ screen, and enough battery life for seven hours of play while being smaller than the original Game Boy!"

The seven hours requirement is so it can survive a long plane trip, something modern handhelds can't quite handle. "I love retro gaming and wanted something to do while travelling," Chris explains. "I didn't want to have to bring a case full of game cartridges or AA batteries with me everywhere, and wanted it all-in-one and easy to use for anyone that would play it."

The idea came to him after learning about RetroPie, and he began to tinker with the Raspberry Pi. The whole thing is custom-made, and it starts with a Raspberry Pi 2 and a 3D-printed case. The case takes the Game Boy as its base inspiration, adding more face buttons as well as shoulder buttons to accommodate PlayStation (and other console) games. There's a 3.5″ screen, a 6000mAh portable battery, amps, speakers, jacks, LEDs, and various buttons and

The buttons on the rear are very basic, but Chris is unsure how to improve them

> "Enough battery life for seven hours of play while being smaller than the original Game Boy!"

switches cannibalised from a NES and several Game Boys.

"It was complex to get everything to fit and work together," Chris admits. "I designed the housing in SolidWorks, wrote programs for the Pi in Python, programmed the AVR in C, designed custom circuit boards, lots of breadboard time, some talking to Adafruit to see what their products can and can't do, etc. But it was relatively easy to build."

The result works well enough for Chris, with power on and off switches, games loading from a USB port on the side, and the ability to charge the battery from the bottom of the case with a micro-USB cable.

Right now, Chris is making a second one for his girlfriend and will eventually write up a full rundown on how to make your own, including slightly improved plans for the circuit boards. Soon you'll be able to make long trips much more interesting.

THE MECHANICAL PROCESS

>STEP-01
On standby

The power switch is controlled by an ATtiny chip that stays in sleep mode until the power is switched on, which then wakes it up so it starts to provide power to the screen.

>STEP-02
While it's on

The Raspberry Pi has a piece of code that lets the ATtiny know that it is on, and it boots straight to RetroPie. The power switch begins a shutdown procedure.

>STEP-03
Turning off

When the ATtiny is told the Pi is shutting down, it turns off the screen and grounds the power, turning off the regulator before going back into sleep mode.

MARK BUTTLE

Mark is a software engineer from Shropshire with a passion for woodworking and electronics – combining both in this project.
bit.ly/1hZFUmu

Quick Facts

- The project started as a challenge from a colleague
- This was Mark's first Pi project
- The controller wires terminate in aircraft connectors
- The power supply comes from an LED lighting set
- A buffer prevents leakage current triggering end-stop relays

PI-POWERED CNC MACHINE

How do you combine a passion for beautiful woodworking with an interest in electronics? **Mark Buttle** did it with a Pi-powered CNC machine...

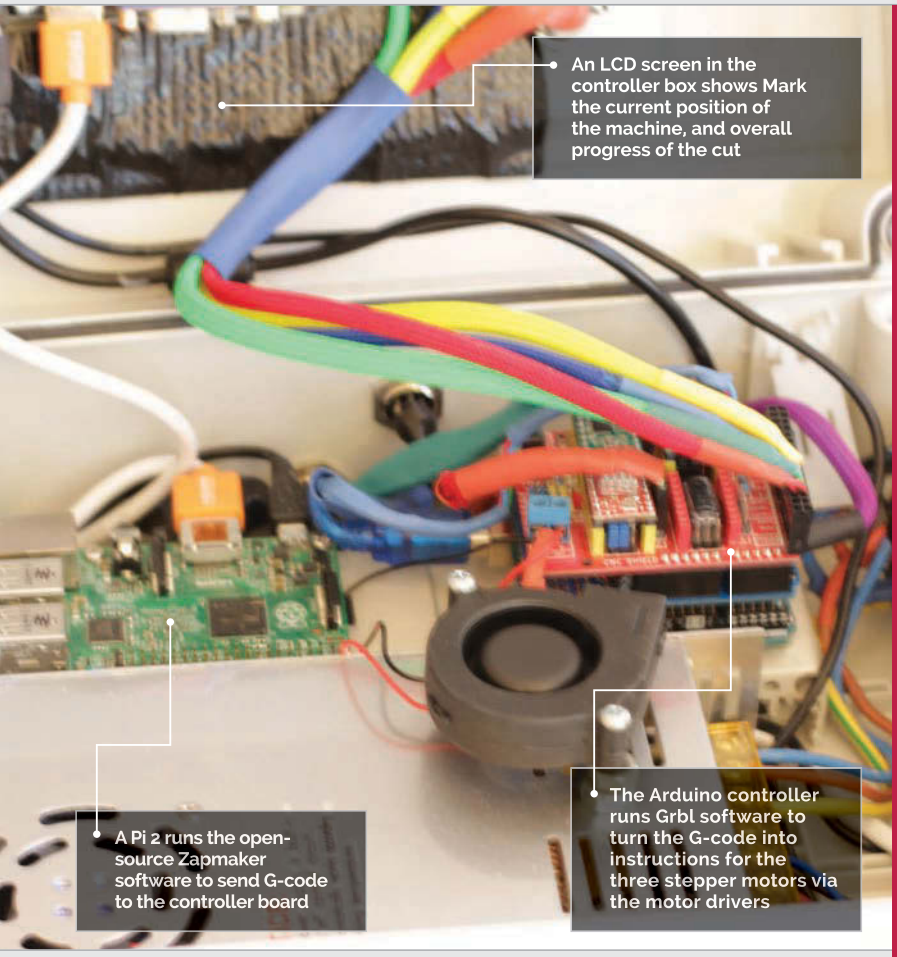
Rural Shropshire, one of the coldest and wettest parts of England, is home to many makers and inveterate tinkerers, with plenty of time holed up in the maker shed, allowing genius to ponder and grow. In fact, the county has a history of makers and thinkers, being the place of origin both of Charles Darwin, and of the industrial cradle of Ironbridge.

Mark Buttle is an engineer who loves to work with wood. His home is full of lovingly handmade furniture, as well as quirky items like a fridge-freezer masquerading as a traditional red telephone box (naturally, there are plans for a familiar blue police telephone box, too). But Mark is a software engineer by day, and also interested in electronics, so is happy to mix technology with traditional crafting skills.

Mark had originally bought an Arduino as a birthday present for a colleague and, intrigued by the board's possibilities, had then bought one for himself. The colleague set a challenge, to see who could come up with the most interesting project, and Mark began looking into building a Computer Numerated Control (CNC) machine for woodworking.

It soon became apparent that the Arduino, while ideal for controlling the stepper motors, would need some extra help, and Mark certainly didn't have time to write all of the software himself. Research online turned up both Grbl – the open-source controller code that runs on the Arduino – and the Zapmaker controller software, which is written for the Raspberry Pi.

"I'd wanted a Pi for a while," Mark tells us. "This is the first



thing that I've done with it. It brings together my passion for electronics and for woodworking."

The build

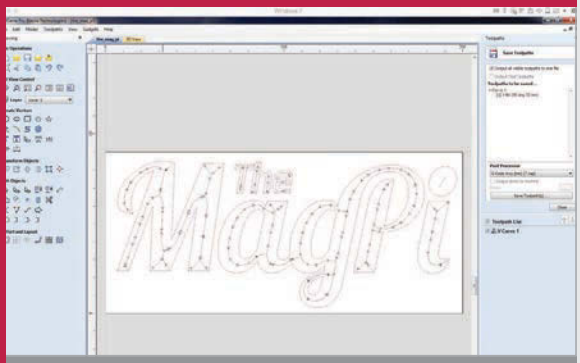
"I've made the complete CNC out of marine ply, all handcrafted, and mounted a DeWalt router to do the cutting," Mark reveals, "using the Raspberry Pi 2 as the controller, which runs a bit of software called CNC Zapmaker, which passes the instructions in G-code to an Arduino Uno. This runs a version of Grbl controller, which has a CNC shield connected to it that drives the three X Y Z stepper motors."

"This is just an overview," admits Mark. "The controller and machine took about two months to build: mainly evenings." In that time, Mark also had to build the physical apparatus of the cutting machine – a 1000mm long, 500mm wide, 700mm high framework, with tracks for clamping the wood, trapezoidal bars for the stepper motors to drive across each axis, and a lot of cabling.

With 200 steps per revolution (on the current setting), the motors allow precision control of movement. One revolution moves the cutting tool 2mm, so one step is just 10 micrometres! – with one thou (a thousandth of an inch), the traditional measure of fine tolerances in engineering, equalling circa 25 micrometres, this is extremely precise for woodcutting. Theoretically re-gearing the stepper motors could improve this again, but that fine a level of detail would be lost against the size of the grain of the wood.

The stepper motors – around £12 each from China – have a power of 107oz holding torque; add in Mark's use of trapezoid rods and heavy-duty fixing bars, rather than the threaded rod and drawer rails used in budget CNC home-builds, and we have a strongly engineered machine that's built to last. Yet, including all the electronics as well as the other hardware, the build price was around £400. A decade ago, you'd find similar

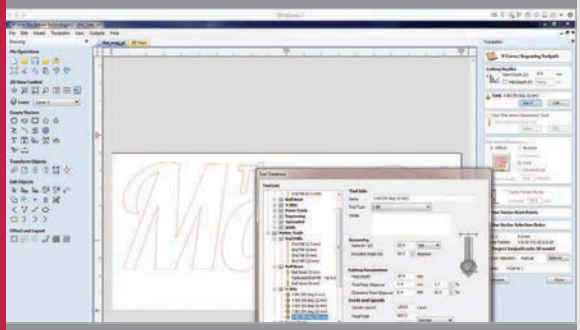
RASPBERRY PI CNC IN ACTION



>STEP-01

At the drawing board

The first task is to work through the design in your choice of CAD software, then let it generate the G-code needed. FreeCAD users will need a CAM plug-in or HeeksCNC.



>STEP-02

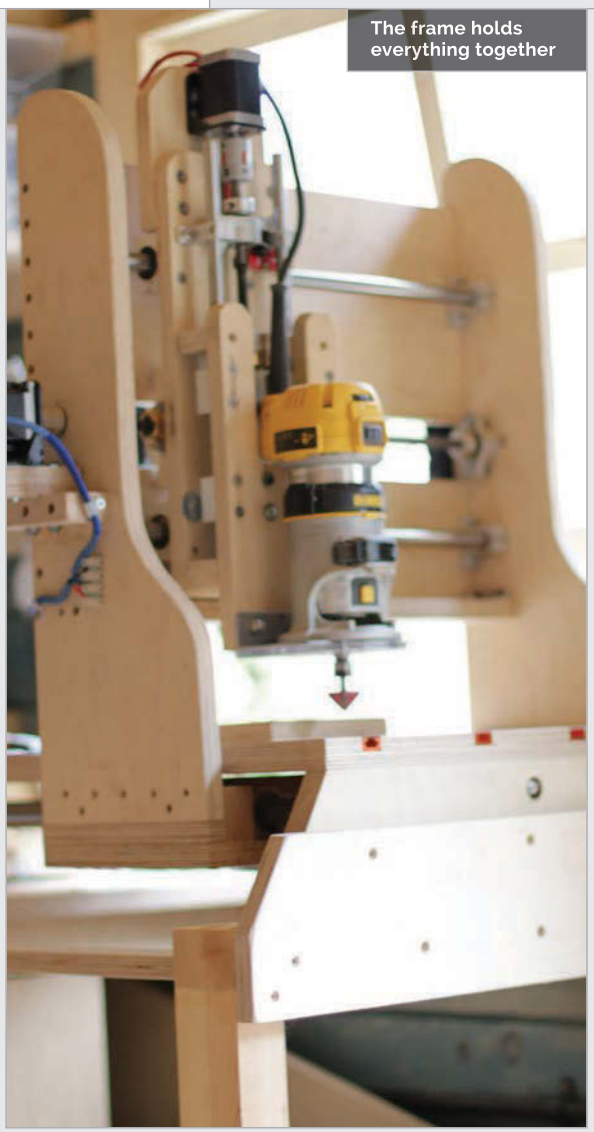
Grbl

Mark uses AutoCAD, which enables him to select the tool bit used, as well as plan the cutting path. The recently open-sourced Blender CAM is one possible alternative for this.

>STEP-03

MagPi meets timber

The Pi creates the Grbl code used by the Arduino, which handles putting the router in the correct places to turn our 3D design into a sculpted or carved piece of wood.


The frame holds everything together

commercial machines costing well into five figures – these are great times for makers. Indeed some of those obsolescent commercial CNCs are being repurposed with homebrew controllers by those without the time to follow Mark's total DIY route.

In control

In the controller, a touchscreen interfaces to the CNC controller, running on a Pi 2 with Zapmaker software: on screen, the control software's grey, pleasingly industrial panel complements the electrical box used as a casing. The Pi communicates with an Arduino Uno board, which uses Grbl to control the stepper motors. The Grbl handling of the Arduino is quite sophisticated – you can send a one-line command such as 'a circle to here, of this diameter,' and it will work out the cuts for you.

The power supply is a heavy-duty 24V unit from some LED lights. 12V and 5V step-down transformers provide power for the LCD screen, Raspberry Pi, stepper motors, Arduino, and shields. The shield controlling the motors can become very hot, and so a small fan is used to cool it.

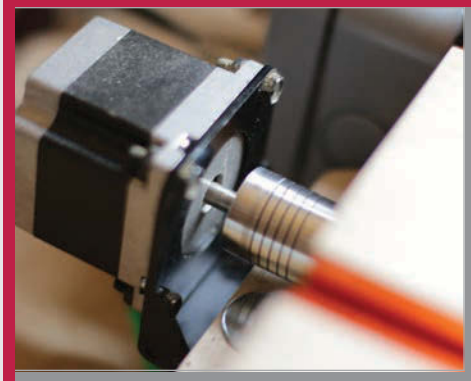
Eagle-eyed readers will have spotted the extra board between the Arduino and the controller shield – this is a buffer board installed after the early prototype kept switching off: "Someone said you need a filter between the Arduino and the CNC controller board," says Mark. A filtering board stops current bleeding through from the motors to the relays for the end stops and switching off the device. This has fixed the problem: "I've checked the voltage drop now on the end stops; it stays at a constant 5 volts."

Making the cut

Before use, G-code is needed to give the coordinates for cutting. The logo in the pictures was converted into a black and white image, then into vectors – as well as optimising the cutting path, and the correct cutting head. FreeCAD is used by many CNC makers, but needs some form of CAM plug-in to generate the G-code and the cutting path.

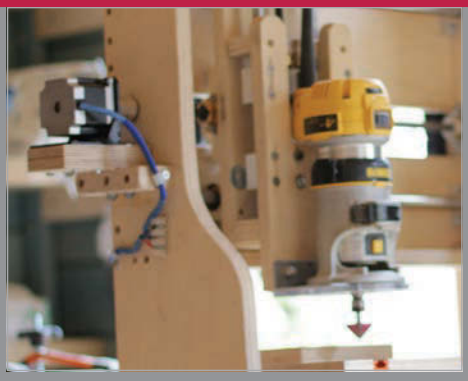
The G-code is just lines of coordinates. The CAD/CAM software has the job of generating it, together with the cutting path that makes the most sense – a CNC machine's version of computing's famous travelling salesman

BUILDING A PI CNC MACHINE



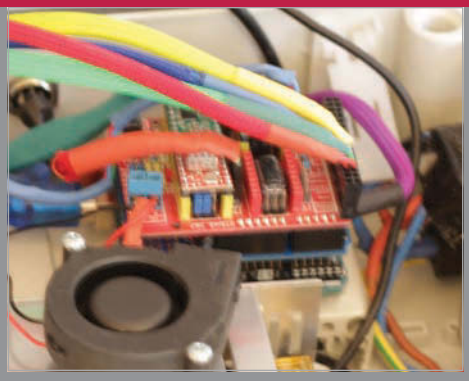
>STEP-01
Stepper motors

Stepper motors – dividing each 2mm revolution of the trapezoidal bars into 200 steps – position a DeWalt router in the X, Y, and Z planes, with an accuracy of 10 micrometres.



>STEP-02
In the frame

Hand-cut from 18mm (¾") marine ply, the frame rigidly holds the trapezoidal bars, stepper motors, and router, as well as keeping the work in place with clamps in the built-in tracks.



>STEP-03
Arduino

An Arduino Uno runs Grbl software to accept cutting instructions in G-code and turn them into electrical pulses to control the stepper motors and position the router precisely in three planes.

problem to find the optimal route between multiple destinations.

Mark is a long-time user of AutoCAD. While it's an expensive option for anyone buying just for a CNC machine, it readily provides everything you need if you're already a user: "On my Mac, I design any shape or carving and convert the 2D or 3D model into G-code, then Wi-Fi it or USB-stick it to the Raspberry Pi. Using the touchscreen, I start the controller up and off it goes."

There are also tripper switches at the end of each trapezoidal guiding bar, to stop the router should it ever be sent to the end of the bar. These were what was being tripped by the current leak before the buffer board was added to the Arduino.

The controller box has four buttons by the screen: on/off, panic!, restart, and emergency stop. It can cut into a 60mm depth of wood. As an alternative to a complete cut, you can optionally set up the controller to leave tabs in, to make it easier to lift out pieces together from the bench.

> "It brings together my passion for electronics and for woodworking"

Well, not quite: safety features are an essential part of any CNC machine. After lining up the cutting tool to the wood block, locking the wood in with retaining bolts set into the tracks, and manually correcting the height, the safety must be clicked on the controller. There's also an emergency stop button on the front of the machine's base. Any powered cutting tool must be built with ways to quickly stop it, just in case of something going wrong.

Looking at *The MagPi* logo in the photo (and on Mark's YouTube channel – **bit.ly/1hZFUmu**), you may be able to see the grooves on the side, matching different depths of cut. This was cut in three passes, taking around five minutes, to allow for the quality of the router bit, which cost only £1.20. Given a better bit, the logo could have been cut in one pass, taking less than a minute-and-a-half.

WHAT NEXT?

Mark tells us he did much of the CNC machine's design ad hoc: "I hand-cut the frames, testing out sizes as I went." This is apparently a contrast to his usual way of working. "Normally I like to have everything carefully thought out from the start."

Should he wish to build another, "Now I can do a template for it to cut a new version of itself." Future plans may include alternatives to Zapmaker: "There are other things I'd like it to do with it. I just like to try things my way." There's also the possibility of similar mechanics and controlling apparatus for a different head: powering a 3D printer, possibly, or to rig up to a Blu-ray laser to try as a cutter (suitably shielded, of course).

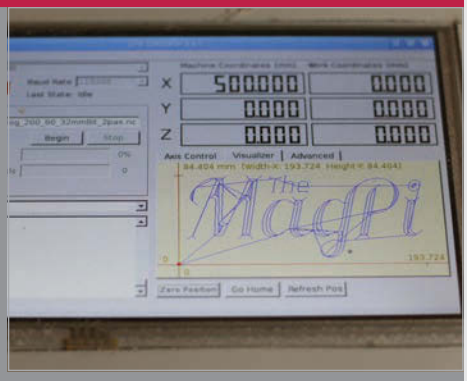
"I like to build," says Mark, which seems to sum things up quite well for many in the maker movement. Having seen some of Mark's other builds – from oak drawers that convert into a Sony PS3 racing car controller, to the famous telephone box refrigerator – we're sure that whatever follows will be an interesting project, as well as something extremely well engineered.

>STEP-04
On the Pi

Zapmaker's Gbrl controller software was originally optimised for a Pentium III with 256MB of RAM, and readily made the transition to the Pi Model B, hardly troubling the Pi 2 in Mark's CNC controller box.

>STEP-05
Raspbian

Zapmaker's CNC software runs on a standard Raspbian install, and the **zapmaker.org** website has full instructions for those new to the wonderful world of Raspberry Pi, as well as Grbl information.



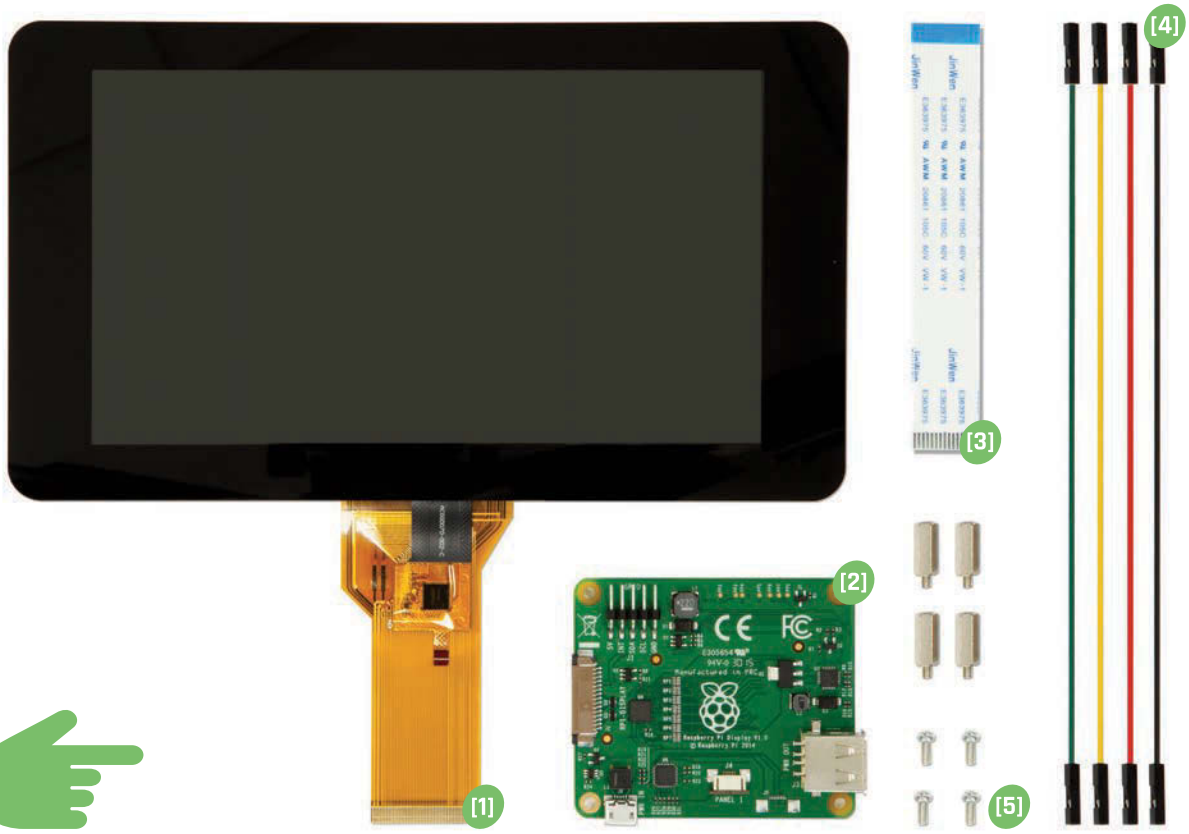
>STEP-06
Grbl controller

Put it all together and the screen shows the cut to be made, waiting for you to hit the Start button after you've secured the block of wood safely in place.

SCREEN TEST

GET TO GRIPS WITH THE OFFICIAL 7-INCH TOUCHSCREEN DISPLAY

In this eight-page special, we look at how to get started with the official Raspberry Pi touchscreen, before trying out Kivy and exploring possibilities for projects...



[1] The touchscreen features two ribbon cables that must be connected to its video board

[2] This board converts the Pi's DSI video signal to a DPI one for the touchscreen

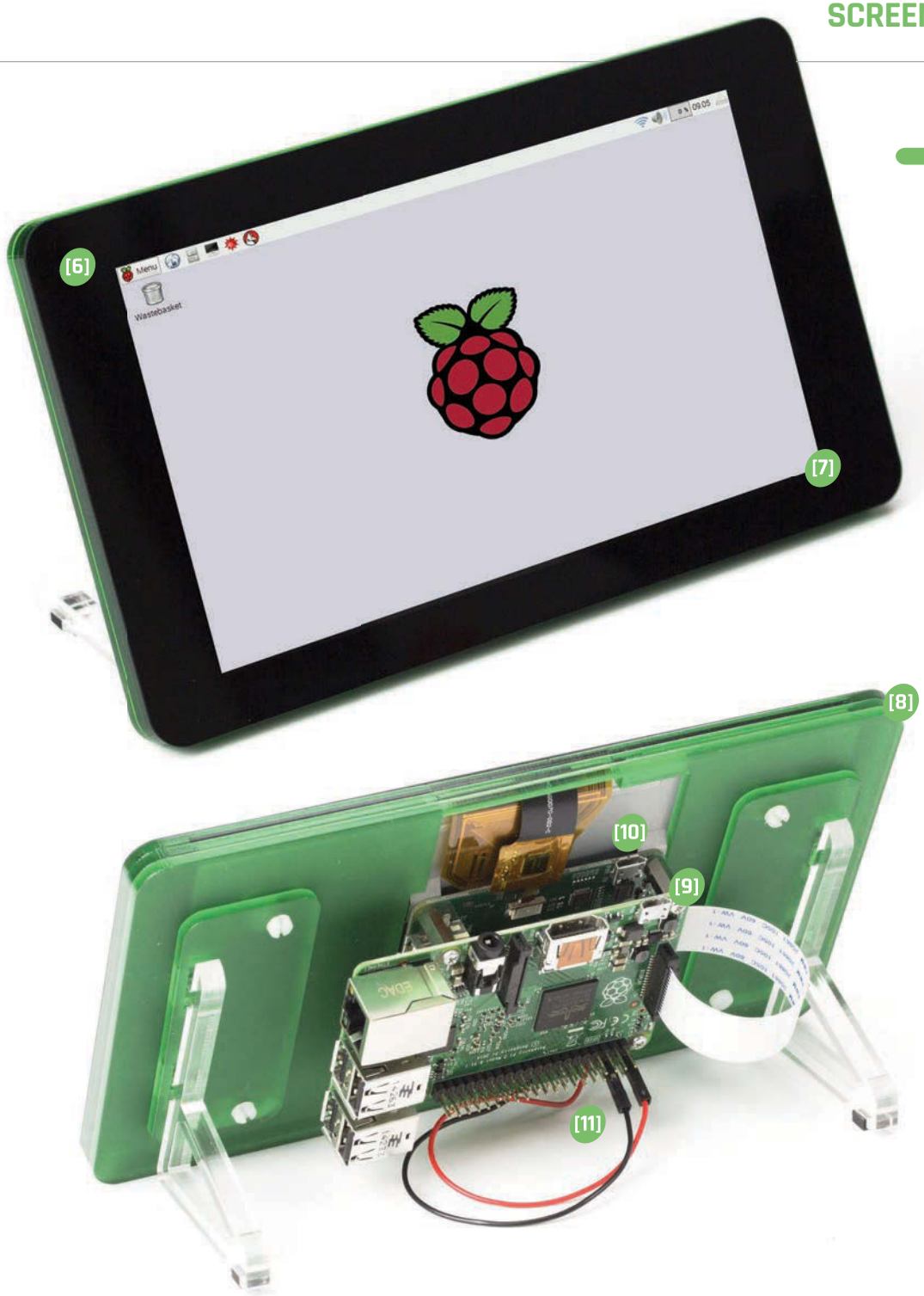
[3] The white ribbon cable is used to connect the screen's video board to the Pi's DSI port

[4] Jumper cables are supplied, which can be used to power the Pi from the board via GPIO pins

[5] Four standoffs and screws are used to attach the board and Pi to the rear of the screen

First mooted over two years ago, the official Raspberry Pi touchscreen display has finally arrived. "I honestly believed it would only take us six months from start to end," says Gordon Hollingworth, the Foundation's director of engineering, "but there were a number of issues we met." Not to mention being diverted by other products such as several new models of the Raspberry Pi itself, of course. Still, it's been well worth the wait and the new 7-inch screen is already proving a huge sales success: at the time of writing, there's a three-week wait for new orders to be shipped by the official Raspberry Pi Swag Store (**bit.ly/1LZNCaA**).

If you're lucky enough to get your hands on one, you'll soon see why. Out of some impressive tech specs, most noteworthy is support for ten-point touch input, which allows for actions such as drag-and-dropping, rotating, and pinching to zoom. The build quality is equally remarkable: Gordon tells us that this is a highly durable "industrial-quality display" (made by Inelco Hunter) that can withstand serious vibration and operate in temperatures ranging from -20° to



[6] The 7-inch LCD display features an 800×480 resolution and 140° horizontal viewing angle

[7] The latest version of Raspbian includes drivers for the screen to enable ten-point touch input

[8] This screen is mounted in Pimoroni's optional stand, which comes in a range of colours

[9] The Raspberry Pi is stacked on the rear of the video board, screwed into the standoffs

[10] The screen is powered via a micro-USB port on its video board, so it can use a standard Pi power supply

[11] The Pi can be powered separately or share the screen's supply via a USB cable or jumper leads (as shown in this photo)

> "The screen comes as a two-part kit comprising the LCD display itself and a daughter video board"

70°C. It's also the first touchscreen to make use of the Pi's DSI (display serial interface) port; this frees up the GPIO header and HDMI port, so the latter can be used by certain applications for external video output.

The screen comes as a two-part kit comprising the LCD display itself and a daughter video board. The video board fits on the display's rear metal backplate with four standoffs, to which the Raspberry Pi itself can be attached. The assembly process (see 'Easy assembly' box on page 40) is very straightforward, however, merely requiring a trio of ribbon cables to be attached between the display, video board, and Pi.

EASY ASSEMBLY

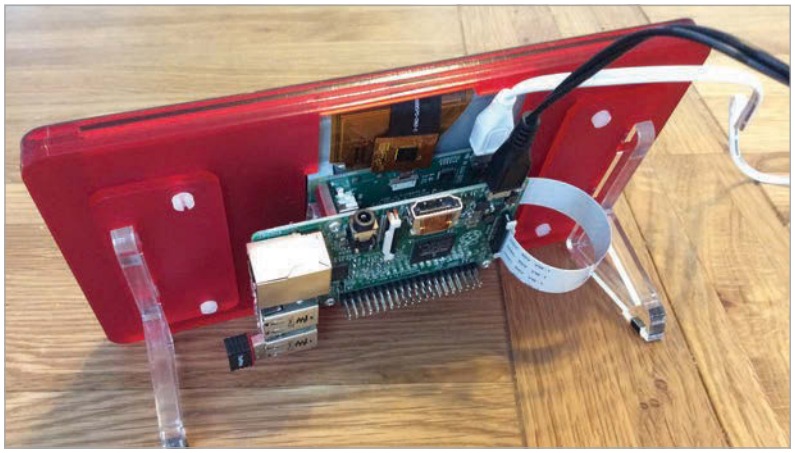
Here's a quick guide to assembling the screen and connecting the Raspberry Pi. Note that the original Pi Models A and B are not currently supported. See **bit.ly/1KOPxi2** for more detailed step-by-step instructions. Note: To easily insert ribbon cables, first pull out the plastic protector from the port; once the cable is inserted, close the plastic protector again to secure it.

Connect the screen's wide ribbon cable to the port on the rear of the video board.

Lie the board on the back of the display and connect the small ribbon to the J4 port.

SCREEN TEST

Once assembly is complete, there are three possibilities for powering the screen and attached Raspberry Pi...

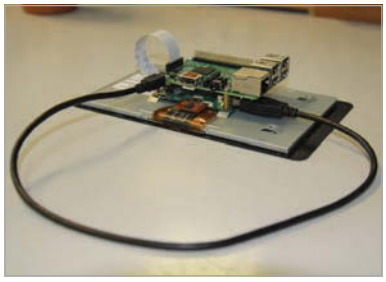


1. TWO SEPARATE POWER SUPPLIES

You can power the screen and Pi separately via their micro-USB ports, using two mains power supplies to ensure you get ample juice for each. Pimoroni is now selling a splitter cable to route power from a single 2A mains supply to both micro-USB ports (**bit.ly/1VWsr1c**).

2. JUMPER LEADS TO GPIO

Using two jumper cables, you can power the Pi from the screen's video board by connecting its 5V and GND GPIO pins. The downside is that this prevents a HAT add-on board from being attached. You'll need a good 2A mains supply (such as the official one) connected to the screen.



3. USB TO MICRO-USB CABLE

While this is possibly the most convenient method, you'll need a 28/24AWG cable (used to charge tablet devices and larger phones), since it has thicker power wires than the more common 28/28AWG type. As with method two, a good 2A main supply is required.

PORTABLE PI

For a portable Pi, you'll need battery power. For this, we'd recommend using one of the many 'power banks' available for powering smartphones and tablets. You'll need one with at least a 2A USB output for a shared power supply, or two outputs – such as the Mi Power Bank 16000mAh (**mi.com/en/pb16000/**) – to power the screen and Pi separately. Battery power opens up the possibilities for creating a 'PiPad' tablet device, although you'll want to enclose the rear electronics somehow: it's possible to build a Pibow case around the Pi, although you'll still need something to protect the video board. Alternatively, you could go for a more radical solution (see 'Reverse mounting' box).

Screw the four standoffs intnto the metal backplate: be careful not to overtighten!

Attach the supplied white ribbon cable to the port at the side of the video board.

Screw the Raspberry Pi to the standoffs, then attach the ribbon cable to the DSI port.

Choose a power method to supply the touchscreen and Pi and you're ready to roll.

TAKE A STAND

Unless you're fitting the touchscreen into a box or case for a project, you'll want a stand for it. Pimoroni sells an easily assembled stand in a choice of colours (**bit.ly/1VrmIuR**). Some generic tablet stands may also be compatible. Alternatively, you could opt to 3D-print your own: designs already available include Alex Eames's raspberry-shaped stand (**thingiverse.com/thing:995394**) and Christopher Masto's wall bracket (**thingiverse.com/thing:1034194**). Or, if you're good at DIY, follow the example of Jimmy Nugent (**bit.ly/1O8xBRD**) and craft a wooden stand!

REVERSE MOUNTING

Upon electing to use an old 3.5″ external hard drive case to enclose his screen setup, the Foundation's Clive Beale discovered that it wouldn't quite fit. His solution was to reverse-mount the Pi (**bit.ly/1L5Uh45**). After a little tweaking, and twisting the DSI cable, it fitted… apart from a small overlap of the screen bezel. Sadly, when Clive tried to trim it with a glass cutter, it cracked! Still, as long as you're careful, reverse-mounting is a good way to radically reduce the depth (to just under 20mm) so you can use a custom case to create a tablet-style device.

STARTING UP

Now you've got your touchscreen assembled, connected, and powered, it's time to start using it...

First off, you'll need to update the Pi's Raspbian operating system to ensure the required touchscreen drivers are installed. Note that while the screen will work with the new Raspbian Jessie edition, we came across a couple of issues regarding the Florence on-screen keyboard and right-click emulation. So, for now, it may be best to stick with Raspbian Wheezy. Once you have logged in, you'll need to update its packages with the following two commands:

```
sudo apt-get update
sudo apt-get upgrade
```

Then reboot the Pi:

```
sudo reboot
```

Another thing you'll want to do when using Wheezy is to get it to boot straight to the X Window GUI desktop. To do this, go to the raspi-config menu (**sudo raspi-config**) and select option 3 (Enable Boot to Desktop/Scratch; at the next screen, select 'Desktop Log in as user 'pi' at the graphical desktop'. Choose to Reboot and the Pi should now boot straight to the desktop when switched on.

Since the touchscreen driver outputs both standard mouse events and full multi-touch events, and therefore can work with X as a mouse, you should now be able to navigate the GUI using touch input. One thing missing by default is a right-click option, but with a bit of config tweaking (see 'Right-click trick'), this can be emulated via a long press of the screen.

On-screen typing

Now, the other obvious problem, unless you have a physical USB keyboard attached, is the question of how you enter text on the touchscreen. Fortunately, there are a couple of virtual on-screen keyboard options readily available in Raspbian: Florence and Matchbox. Just use the following terminal commands to install one or both of them:

```
sudo apt-get install florence
sudo apt-get install matchbox-keyboard
```

Florence will then appear under the Universal Access menu, while Matchbox appears under Accessories, as 'Keyboard'; if it doesn't appear straight away, try rebooting. You may want to add a menu shortcut to toggle it on and off (**bit.ly/1VVTcgQ**). Either keyboard can be easily repositioned and resized by dragging the top bar or bottom-right corner respectively. By default, Florence is always shown on top of anything else on screen; to set this option for Matchbox, tap the top-left box, then Layer>Always on top. Both can also be rolled down when not needed, to free up screen real estate.

Of the two, Florence has the greater range of settings, including a choice of colour schemes, shapes, and fonts. However, as noted, we had problems using Florence in Raspbian Jessie: every time we pressed a key, it disappeared. Florence works fine in Wheezy, though, albeit with some keys missing labels.

Audio and video

When it comes to audio, the official touchscreen lacks built-in speakers, so you'll need to use the Raspberry Pi's A/V 3.5mm jack (on Models 2 B, B+, and A+) to output audio to some external PC speakers or a hi-fi system. One thing to note is that the Pi's jack is wired differently from some, so you may find that left/right channels are swapped.

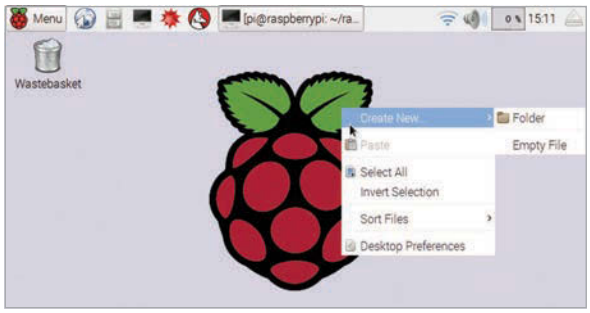
One intriguing possibility is the potential to simultaneously output video to a TV or monitor via the Pi's HDMI port, for a dual-screen setup. Currently, it's not possible to output Raspbian's X Window GUI to both touchscreen and TV. However, some applications already have support for a secondary video output, including OSMC (for the Kodi media centre) and omxplayer. For instance, in omxplayer, you can output a video to the touchscreen with the default command:

```
omxplayer video.mkv
```

...and another via HDMI using:

```
omxplayer --display=5 video.mkv
```

With a bit of tweaking, it's also possible to run a Kivy touch interface on the touchscreen while the X Window GUI is output to an HDMI TV or monitor. You can find out more about using Kivy to control physical devices on page 44.



RIGHT-CLICK TRICK

While the X Window GUI was never designed to work with a touchscreen, there is a handy tweak to get it to emulate a right mouse click when you long-press the screen. Note: at the time of writing, we could only get this to work in Raspbian Wheezy and not Jessie.
From a terminal window, create a new config file with:

```
sudo nano /etc/X11/xorg.conf
```

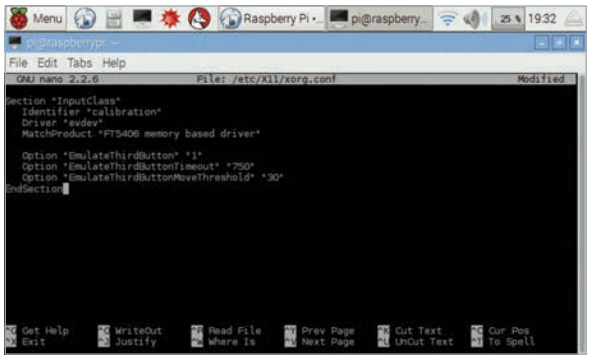
Then add the following code to it:

```
Section "InputClass"
    Identifier "calibration"
    Driver "evdev"
    MatchProduct "FT5406 memory based driver"

    Option "EmulateThirdButton" "1"
    Option "EmulateThirdButtonTimeout" "750"
    Option "EmulateThirdButtonMoveThreshold" "30"
EndSection
```

Press **CTRL+X** to exit, then **Y** and **ENTER** to save the file. Now reboot the Pi:

```
sudo reboot
```

You should find that long-pressing on the desktop brings up the standard right-click menu. So whenever you want to right-click on something, simply long-press it.



F.A.Q.

We tackle the most common issues with the official touchscreen display...

Help, I've got a black screen!
Check that the DSI cable is pushed firmly into place. You can also try reversing it, by switching which ends go to the video board and Pi.

I've got a white screen!
This is likely due to the screen's ribbon connector not being seated correctly in the video board.

It's stuck on the rainbow test screen.
This means the Pi isn't getting enough power to boot up; this issue occurs most commonly when powering it from the video board via USB. Try a different cable, or use the jumper wires method. Also, note that even if the Pi does boot up, a permanent rainbow square in the top-right corner indicates undervoltage.

My screen is upside down!
If using an attached stand, such as Pimoroni's, you may find that the latest Raspbian update – which flipped the display output for a better desktop viewing angle – results in the screen being upside down. To correct it, you need to add a line to the **config.text** file. Open it with:

```
sudo nano /boot/config.txt
```

...Then add the line:

```
lcd_rotate=2
```

Press **CTRL+X** to exit, then **Y** and **ENTER** to save the file. Now reboot the Pi and the display should be the right way up!

Touch input isn't working, or stops working.
Assuming that you've checked all the connections are fine, this is likely down to a bug in the touchscreen that's been fixed in the latest version of Raspbian. To sort it out, use the following commands:

```
sudo apt-get update
sudo apt-get install
--reinstall libraspberrypi0
libraspberrypi-{bin,dev,doc}
raspberrypi-bootloader
sudo reboot
```

Some windows are too big for the screen and I can't see the right/bottom edges.
This may be due to some developers assuming a minimum screen resolution of 1024×768 pixels. Right-click (using a mouse or long-press emulation trick) on the top of the window, select Move from the menu, then use the arrow keys to reposition the window to reveal the hidden parts.

I have black borders around the desktop - it's not using all of the display.
This may be because you have overscan enabled in the bootup config file. Comment it out.

I want to use my screen in portrait mode, like a smartphone, by rotating it 90 degrees.
There is a workaround to enable this, involving rotating both the display and touch input. See **bit.ly/1Gklqul** for details.

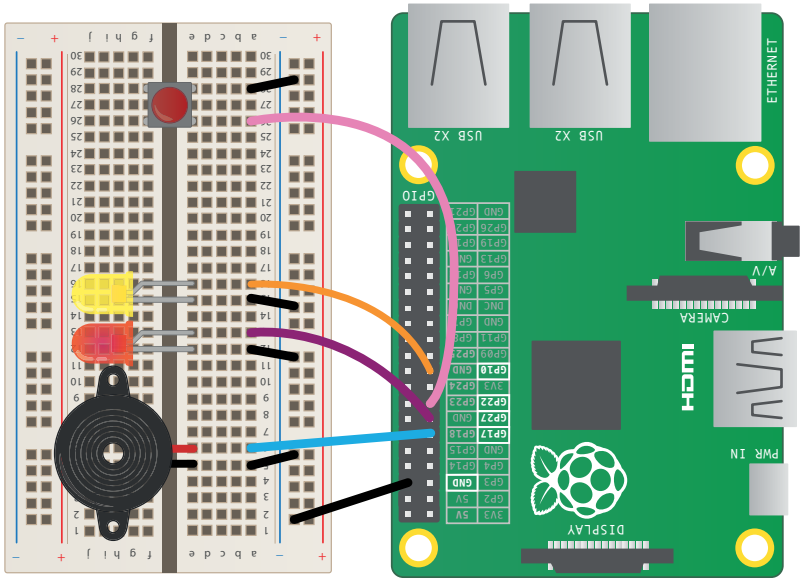
TAKING THINGS FURTHER

We show how to install Kivy to create a custom touch interfaces, and look at some possible projects for your new screen...

So, now that you've got your Pi and touchscreen up and running, what are you going to do with it? While you could simply use it as a portable Pi, there are countless possibilities for projects. Raspberry Pi's US product evangelist Matt Richardson tells us, "Many Raspberry Pi projects need a user interface, but these projects can't always accommodate a full monitor, keyboard, and mouse. The touchscreen can offer these capabilities without needing extra components." This is where Kivy may come in handy: it's a framework for Python which allows a developer to create graphical on-screen interfaces. "It's a modern framework," says Matt, "so it works well with multi-touch input, which the new Raspberry Pi display provides."

Here we'll walk you through how to set up Kivy using Matt's instructions. We'll assume that you've already got your Pi fully updated using **`sudo apt-get update`** and **`sudo apt-get upgrade`**, and the screen's touch input is working correctly.

Below Follow this diagram to wire up your components



Install Kivy

In a terminal window, open up the Apt sources list:

```
sudo nano /etc/apt/sources.list
```

At the end of the file, add the following line to add sources for Gstreamer:

```
deb http://vontaene.de/raspbian-updates/ . main
```

Press CTRL+X, then Y and ENTER to save the file.

Now download and add the GPG key for the Gstreamer sources with the following commands:

```
gpg --recv-keys 0C667A3E
gpg -a --export 0C667A3E | sudo apt-key add -
```

Note: If you get an error (**`gpg: keyserver receive failed: bad URI`**) after the first command, just try it again.

Install the dependencies with the following two commands:

```
  sudo apt-get update
  sudo apt-get -y install pkg-config
libgl1-mesa-dev libgles2-mesa-dev
python-pygame python-setuptools
libgstreamer1.0-dev git-core
gstreamer1.0-plugins-{bad,base,good,ugly}
gstreamer1.0-{omx,alsa} python-dev
```

You'll now need to install pip from source, as the version in the version in Raspbian's Apt repository is too old. Note: ignore any messages about **`InsecurePlatformWarning.`**

```
  wget https://raw.github.com/pypa/pip/
master/contrib/get-pip.py
  sudo python get-pip.py
```

Install Cython, Pygments, and docutils. While the Pygments and docutils packages aren't required for Kivy, the example code you'll execute uses them. (This step may take quite a while.)

```
  pi@raspberrypi ~ $ sudo pip install
cython pygments docutils
```

Next, it's time to install Kivy globally, with the following series of commands. Again, this can take a long time, so be patient.

```
git clone https://github.com/kivy/kivy
cd kivy
python setup.py build
sudo python setup.py install
```

To generate a Kivy configuration (so we can modify it to enable touch input), run the pictures demo:

```
python ~/kivy/examples/demo/pictures/main.py
```

Now let's modify that Kivy configuration file:

```
nano ~/.kivy/config.ini
```

Go to the **[input]** section of the file and replace the lines in there with the following:

```
mouse = mouse
mtdev_%(name)s = probesysfs,provider=mtdev
hid_%(name)s = probesysfs,provider=hidinput
```

Try it out

Launch the multi-touch pictures demo again:

```
python ~/kivy/examples/demo/pictures/main.py
```

Use multi-touch gestures to drag, rotate, and pinch-zoom the photos. Press **CTRL+C** to exit. To see Kivy's many UI elements, launch the showcase:

```
python ~/kivy/examples/demo/showcase/main.py
```

You can explore the other examples in **~/kivy/examples/** if you like.

Now it's time to try out Matt's demo, linking the Pi's GPIO pins up to some electronic components to give a basic example of physical input and output using Kivy. A piezo buzzer is connected to BCM GPIO pin 17, a red LED to pin 27, a yellow LED (that will flash) to pin 10, and a button has one leg connected to pin 22 and the other to ground (see page 44 diagram).

Unless you're using Raspbian Jessie, you'll need to be root to control the GPIO pins. So you'll need to copy the edited config file to the root account:

```
sudo cp ~/.kivy/config.ini /root/.kivy/config.ini
```

You can now run the code:

```
cd rpi-kivy-screen/
sudo python main.py
```

The left on-screen panel will light up whenever you press the physical button on the breadboard. The middle panel can be touched to toggle the red LED on/off. Touch the right panel to cause the buzzer to emit a short beep. The slider is used to control the speed of the flashing yellow LED.

This is just a simple demo that barely scratches the surface of what can be achieved using Kivy with the touchscreen and GPIO pins. Matt says he expects people to do some really cool stuff with it. Full details, code, and a video demonstration of his Kivy example can be found at **github.com/mrichardson23/rpi-kivy-screen.**

POTENTIAL PROJECTS

As we've seen with third-party screens, there are countless potential projects for making use of the official touchscreen. Matt Richardson tells us, "I foresee people using the display for projects like home control panels. They could be a single point of control for heating and air conditioning, show status of doors and windows, or control home entertainment systems. It wouldn't be difficult to mount the panel inside a wall. With power and a USB Wi-Fi dongle, a home control panel could be fully extensible.

"The display could also be used to control devices like 3D printers, CNC routers, and homebrew sound systems," adds Matt.

Since the official touchscreen has only just been released, it's early days, but we have already spotted a few projects in the wild. If you're working on one yourself, please do let us know.

PiPad 2.0

MCM Electronics' Michael K Castor, creator of the original 'PiPad', has designed and 3D-printed a case for a prototype tablet device using the official touchscreen (**bit.ly/1NfXdxz**). Like Clive Beale, he's reduced the depth of the setup by reverse-mounting the Pi on the screen's video board, bending out two GPIO pins to attach jumper leads for power. "The result is a self-contained and fully functional Raspberry Pi-based tablet that serves for the foundation of the 'PiPad 2.0'," says Michael. If you want to 3D-print the case, the files are available on GitHub: **bit.ly/1VAmeTd**.

Home Hub

One obvious use for the official touchscreen is to use it as a touch control panel for a home automation system. Indeed, the Foundation's Gordon Hollingworth is already using his screen and Pi with an add-on Pi-mote board (**bit.ly/1MtujUU**) and Energenie mains switches to remotely control the electrical sockets in his home.

Carputer

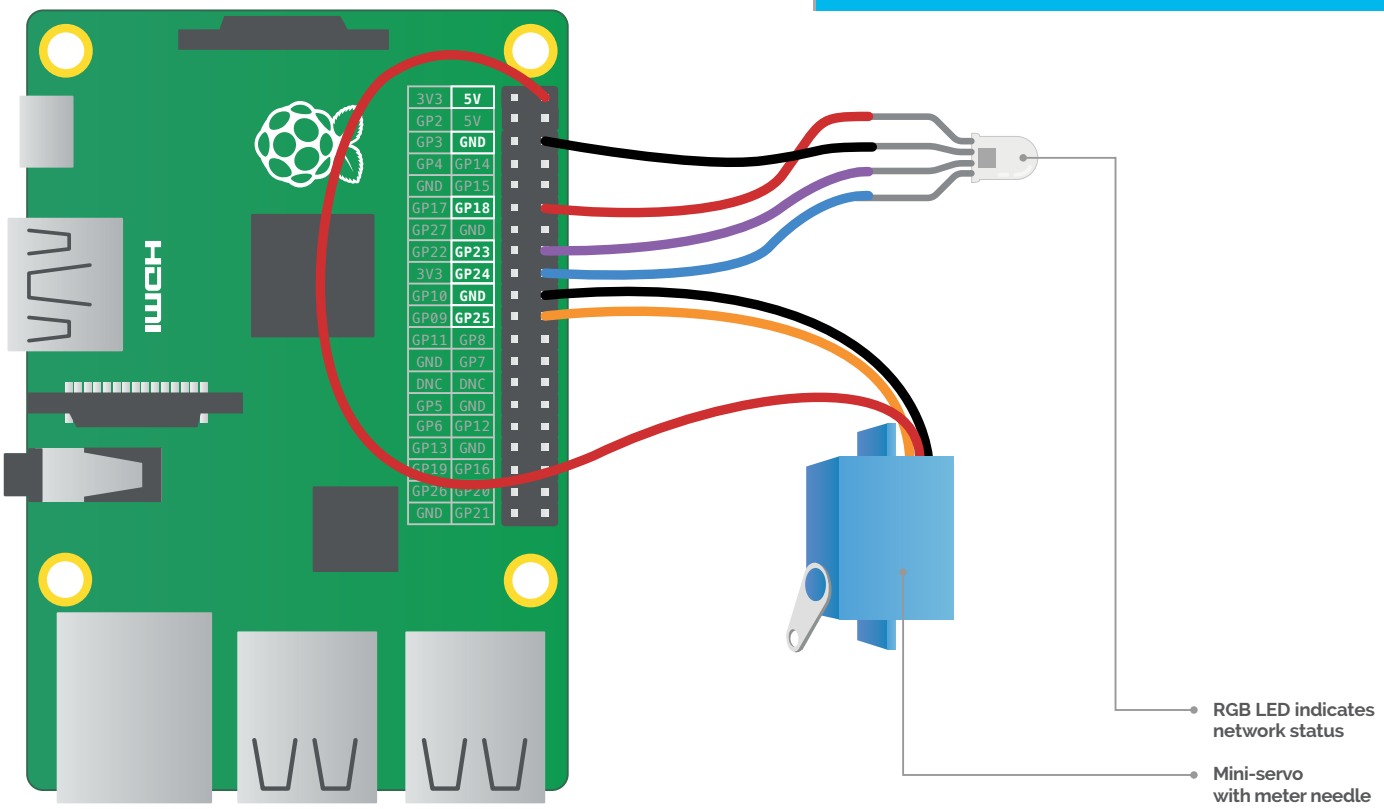
Chris is using his new official touchscreen for a carputer project (you can see it on Instagram here: **bit.ly/1NPGwcy**). He says he's "planning on having Kodi for multimedia and RetroPie for passenger gaming." Both are already up and running on the screen.

EVERYDAY ENGINEERING PART 9

SIMON MONK

Simon Monk is the author of the *Raspberry Pi Cookbook* and *Programming the Raspberry Pi: Getting Started with Python*, among others.

simonmonk.org
monkmakes.com



PINGOMETER

You'll Need

- 9g mini servo motor
- Raspberry Squid RGB LED
- 3× male-to-female jumper wires
- Cocktail stick (to make needle)
- Glue
- Small food container as an enclosure
- Drill and craft knife

Solve real-world electronic and engineering problems with your Raspberry Pi and the help of renowned technology hacker and author, **Simon Monk**

The 'pingometer' gives a visual indication of how much lag there is between your Raspberry Pi and your desired server on the internet. As the Raspberry Pi will be connected to the same network as your PC or games console, this will give you an indication of just how much delay there is between your network and the online game server that you want to play on. It does this using a Linux network command called 'ping' that tells you how long it took for a small packet of data to get to the server and then back to your computer. This value is in milliseconds (thousandths of a second).

As well as a servo-controlled meter indicating the ping time (between 0 and 1,000 milliseconds), an RGB LED will change from green for a fast ping, to orange for an okay ping, to red for a slow ping. If the ping fails to connect to the remote server, then the LED will be blue.

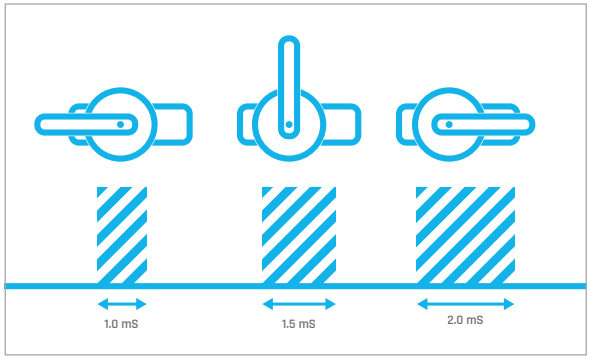
Above The pingometer shows your connection speed

As you'll see from the list of required components on the previous page, this project has both the servo motor and RGB LED module plugged directly into the Raspberry Pi GPIO pins, so no breadboard is needed.

The servo motor is of a type called 9g, and you can find these on eBay for very little money.

The RGB LED is a device called a Raspberry Squid that combines a bright 10mm RGB LED with built-in current-limiting resistors and female header leads. You can find full instructions for building your own Squid on the GitHub page for the Squid's accompanying library (**github.com/simonmonk/squid**). If you would rather buy a ready-made Raspberry Squid, however, you can find one on Amazon or see **monkmakes.com/squid**.

The food container came from a supermarket and has the nice feature that if you leave the top clip undone, it acts as a diffuser for the LED.



Above Controlling a servo motor

Servo motors

The key to this project is the servo motor that moves the meter needle. Servo motors are different from more conventional motors in that they are not free to rotate continuously. In fact, they can only normally rotate about 180 degrees.

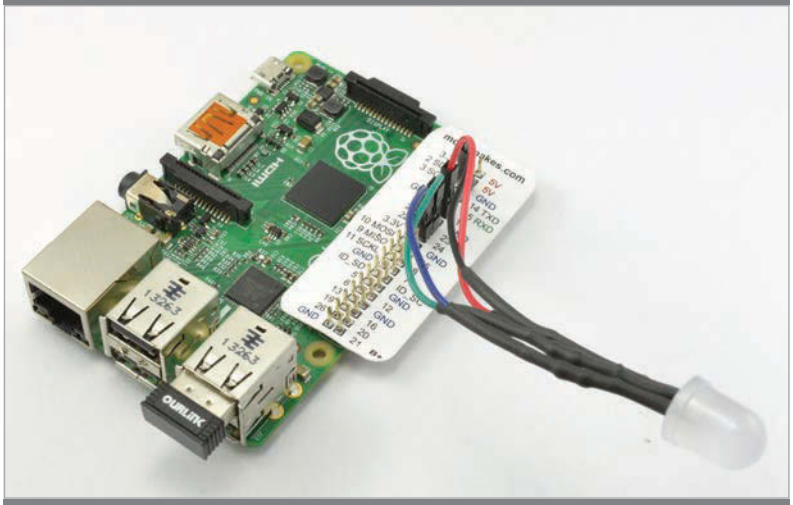
Servo motors have three leads: two power leads (ground and 5V) and a control lead. The control lead is fed a series of pulses of varying length. The length of the pulses will determine the servo motor's angle. If the pulses are around 1 millisecond, the arm of the servo will be at one end of its range – let's call that 0 degrees. A pulse length of 1.5 milliseconds will position the servo arm right in the middle of its range (90 degrees), and a 2 milliseconds pulse will put the arm at the far end of its range (180 degrees). In practice, servos often don't do the full 180 degrees – the range is sometimes more like 0 to 170 degrees.

Building your pingometer

As with all projects, it is a good idea to test the project out and get everything working while the parts are all out on your workspace. Once you know all is well, you can install the project in its enclosure.

BUILDING THE PROJECT

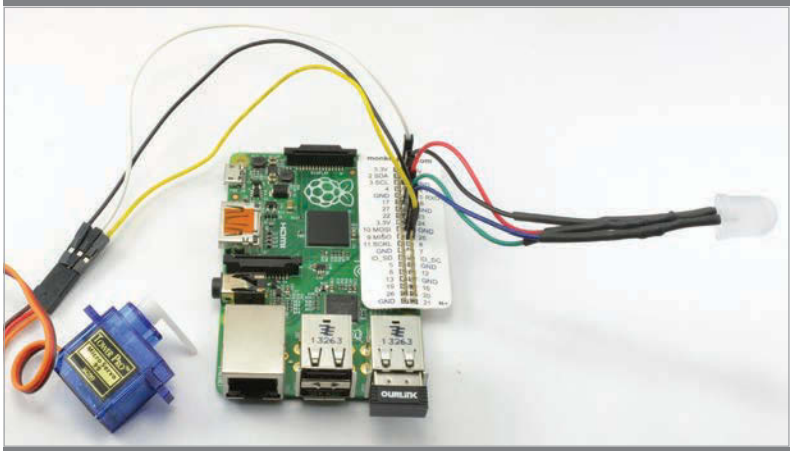
This is a pretty straightforward project to build. Everything just plugs into the GPIO header pins.



>STEP-01

Attach the Raspberry Squid

Connect the black header of the Raspberry Squid to a GND connection of the GPIO header, the red lead to GPIO 18, the green to GPIO 23, and the blue to GPIO 24.



>STEP-02

Attach the servo

When you first plug the servo onto the GPIO connector, there will be a surge of current, and this can be enough to cause your Raspberry Pi to reset. So it's best to attach the servo to your Raspberry Pi while it is powered off.

Use the three female-to-male jumper wires to connect the brown lead of the servo to a GND pin on the GPIO connector, the red servo lead to 5V, and the orange lead to GPIO pin 25.

Fit one of the servo arms loosely onto the motor, just so that you can see the servo moving, and jump ahead for a moment to the software section to try out the project before you go boxing everything up.

>STEP-03

Fix the servo to the lid

Use a hot glue gun, or other glue, and stick the cocktail stick onto the servo arm. You may need to shorten the stick a little to suit the size of your container. Carefully cut out a rectangle in the lid for the front of the servo to push through, and glue the servo in place. Resist the temptation to screw the servo arm firmly into place just yet, as some adjustment will probably be necessary later.

>STEP-04

Finish the boxing

Drill 10mm holes in the box for access to the USB socket and for the Raspberry Squid. If you are using an Ethernet connection, then you will need another hole for that. We used a USB WiFi adaptor. Make sure that everything lines up okay for the servo motor.

In the code downloads for this project you will also find an image file for a very simple scale. You can print this out, cut it down to size, and then stick it on the inside of the plastic food container.

Now that the hardware side of the project is complete, we just need to get the software running. The program is written in Python and uses the Squid library to control the colour of the Raspberry Squid. To install the Squid library, make sure that your Raspberry Pi has an internet connection and then run the commands:

```
git clone https://github.com/simonmonk/squid.git
cd squid
sudo python setup.py install
```

You can download the program for this project from your Raspberry Pi command line, using:

```
git clone https://github.com/simonmonk/pi_magazine.git
```

To run the program, change directory to the one where the code for this project lives and then run the program using the commands below:

```
cd /home/pi/pi_magazine/09_pingometer
sudo python pingometer.py
```

When you run the program, you may find that the needle is in the wrong place. The section on using the pingometer will explain how to fix this.

How the code works

The Python code for this program is pretty heavily commented. You will probably find it handy to have the code up in an editor while we go through it.

The program starts by importing the **subprocess**, **time**, **RPi.GPIO** and **squid** libraries that it needs.

The constant **HOSTNAME** is used to specify the server whose ping is to be measured. If you are using this project for gaming, then put the address of the gaming server that you are going to use here.

The **PING_PERIOD** variable determines how often the ping is measured, and **GOOD_PING** and **OK_PING** determine the colour that the Squid will display. If the ping is less than **GOOD_PING**, the LED will be green; if it's greater than **GOOD_PING** but less than **OK_PING**, it will be orange; otherwise it will be red.

The variable **SERVO_PIN** specifies the pin used to control the servo motor, and **MIN_ANGLE** and **MAX_ANGLE** determine the servo arm's range of movement, as we don't want it swinging through its full 180 degrees.

Next, the **SERVO_PIN** is set up as a PWM pin so that we can generate pulses of the right length to move the servo.

The variable **squid** is initialised using the GPIO pins for the red, green, and blue channels (18, 23, and 24).

The function **map_ping_to_angle** converts a ping time of between 0 and 1 seconds to an angle between **MIN_ANGLE** and **MAX_ANGLE**.

Language
>PYTHON

Pingometer.py

```
import subprocess, time
import RPi.GPIO as GPIO
from squid import *

HOSTNAME = "us.mineplex.com"
# HOSTNAME = "google.com"
PING_PERIOD = 3.0 # seconds
GOOD_PING = 0.1   # seconds
OK_PING = 0.3     # seconds

SERVO_PIN = 25 # control pin of servo
MIN_ANGLE = 30
MAX_ANGLE = 150

# Configure the GPIO pin
GPIO.setmode(GPIO.BCM)
GPIO.setup(SERVO_PIN, GPIO.OUT)
pwm = GPIO.PWM(SERVO_PIN, 100) # start PWM at 100 Hz
pwm.start(0)

squid = Squid(18, 23, 24)

def map_ping_to_angle(ping_time):
    # ping timeout of 1000 ms sets maximum
    # ping min of 0
    # Fast ping needle over to the right
    angle = ping_time * (MAX_ANGLE - MIN_ANGLE) + MIN_ANGLE
    if angle > MAX_ANGLE :
        angle = MAX_ANGLE
    if angle < MIN_ANGLE :
        angle = MIN_ANGLE
    return angle

# Set the servo to the angle specified
def set_angle(angle):
    duty = float(angle) / 10.0 + 2.5
    pwm.ChangeDutyCycle(duty)
    time.sleep(0.2); # give the arm time to move
    pwm.ChangeDutyCycle(0) # stop servo jitter

def ping(hostname):
    try:
        output = subprocess.check_output(
"ping -c 1 -W 1 " + hostname, shell=True)
        return float(output.split('/')[5]) / 1000.0
    except:
        return -1

try:
    while True:
        p = ping(HOSTNAME)
        # p = input("ping=")  # Use for testing
        print(p)
        set_angle(map_ping_to_angle(p))
        if p == -1 :
            squid.set_color(BLUE)
        elif p < GOOD_PING:
            squid.set_color(GREEN)
        elif p < OK_PING:
            squid.set_color((50, 15, 0)) # Orange
        else:
            squid.set_color(RED)
            time.sleep(PING_PERIOD)
finally:
        GPIO.cleanup()
```

The function **set_angle** sets the servo pulse to the correct duty cycle, and so also sets the pulse length to position the servo. To prevent excessive jittering, after generating pulses for 0.2 of a second, the duty is set to 0. This has the effect of the servo arm moving to the correct position and then staying still until **set_angle** is called again.

The code to actually get the 'ping' is contained in the **ping** function. This makes a Linux system call to ping, capturing what would normally appear in the terminal into the variable **output**. This would look something like:

```
round-trip min/avg/max/stddev =
23.739/23.739/23.739/0.000 ms
```

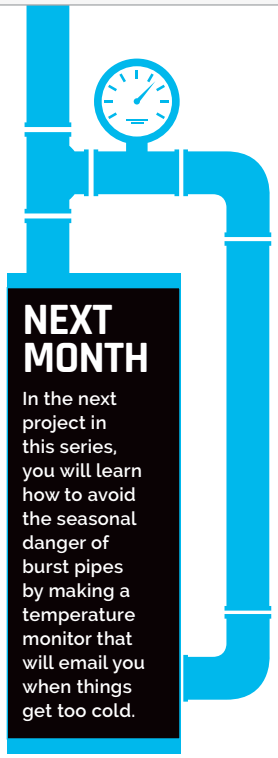
The **ping** command uses the **-c 1** parameter, so only one ping is made and the minimum, average and maximum are all the same value. To extract that value (23.739 in the example above), the string is split into an array of strings separated by '/'. The content of index 5 of this array will be one of the elements containing this number. Dividing it by 1,000 changes the units from milliseconds to seconds, as used in the rest of the program. If the call to **ping** fails for any reason, then the try/catch structure will ensure that the function returns -1.

The main loop repeatedly measures the ping and then sets the servo position and LED colour accordingly.

Using your pingometer

Below the line **p = ping(HOSTNAME)**, there is alternative code for setting **p**, simply by inputting a value. To make it easier to position the servo arm, remove the # from the front of this line and run the program again. Enter a value of 0.5 for **p** and after the arm has moved, remove the arm from the servo and then put it back, but pointing straight up, as this is the central position for the meter. You can then put the # back in front of the line again when you are done.

The settings used here for **GOOD_PING** and **OK_PING** are correct for your expert's somewhat slow network where he lives. You will probably need to change these values to suit your setup.



NEXT MONTH

In the next project in this series, you will learn how to avoid the seasonal danger of burst pipes by making a temperature monitor that will email you when things get too cold.

RICHARD HAYLER

Richard is a mentor at CoderDojo Ham. His school CodeClub was one of the winning teams in the Primary School Astro Pi competition.
richardhayler.blogspot.co.uk
coderdojoham.org

MAGIC PRESENTATIONS WITH SKYWRITER

You'll Need

- Skywriter HAT or board **shop.pimoroni.com**
- Skywriter API library **github.com/pimoroni**
- Python AutoPy library **github.com/msanders/autopy**
- LibreOffice Impress **libreoffice.org**

Amaze your audience with baffling magical powers or mastery of the Force using a Skywriter HAT

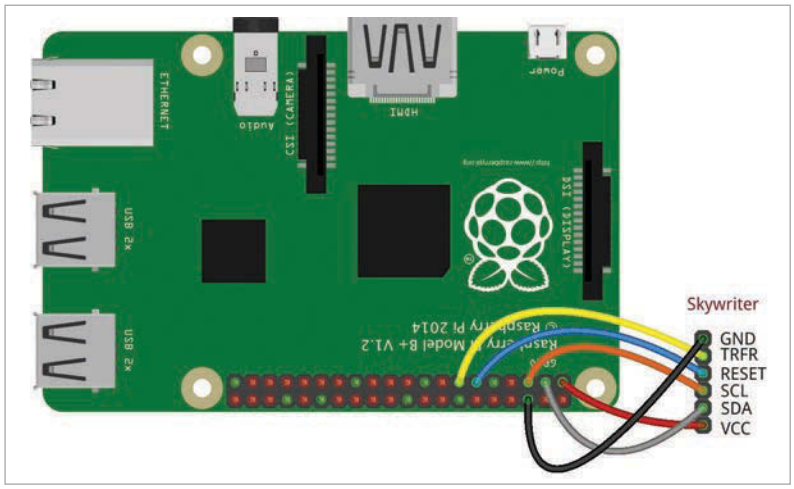
Let's face it: simply using a Raspberry Pi to run the slides for your talk will make the audience think you're pretty cool. But if that's not impressive enough, why not dazzle them further by using your telekinetic powers to flip through the presentation and annotate the slides by drawing in the air? You don't even need to be a graduate of Hogwarts or the Jedi Academy: just get yourself a Skywriter device and some simple Python, and you're ready to rock.

The Skywriter device uses a grid of transmitting electrodes to generate an electric field that propagates around the surface in three dimensions. When you move your hand above the Skywriter, it disturbs this field and these variations are detected by the receiver electrode grid. These measurements can be used to calculate the position and movement direction of your hand.

>STEP-01

Connect the Skywriter device

If you have a Skywriter HAT, this just connects onto the GPIO pins like other HATs. If you have the larger Skywriter board, you'll need to connect six GPIO pins to the matching pins at the top, as shown below.



Below You can use longer wires or a ribbon cable to locate the Skywriter board away from the Pi if needed

>STEP-02

Install the software

Make sure you have the latest version of Raspbian, with all updates installed. As usual, those helpful Pimoroni Pirates supply a single script to handle the installation, including the full Python API. Like most HATs, the Skywriter needs the I²C bus on the Pi to be enabled, so if you haven't already got this activated on your Pi, you'll need to reboot before the Skywriter will work.

```
curl -sSL get.pimoroni.com/skywriter | bash
```

You'll also need the AutoPy Python library and its dependencies, so install these with:

```
sudo apt-get install libx11-dev libxtst-dev
```

...and then:

```
sudo pip install autopy
```

>STEP-03

Test your Skywriter

The Python API has example scripts to help you become familiar with the way Skywriter works:

```
cd Pimoroni/skywriter
sudo python test.py
```

Now wave your hand around in the air just above your Skywriter. You should see three columns of scrolling numbers corresponding to your hand's position in a three-axis (x/y/z) box over the device. The Python library is preconfigured to recognise certain gestures: a flick (swiping over the Skywriter), a tap or touch (bring your hand down to just above the surface), and, trickiest of all, the Airwheel (wiggle a finger in a circular pattern above the Skywriter). It takes a while to get the hang

of reproducing these gestures so that they are always detected, so spend some time practising. You can edit the **test.py** script and comment out the **@skywriter.move()** function (which displays the x/y/z numbers) to make it easier to see when you've nailed one of the gestures.

>STEP-04

Configure your presentation software

Now it's time to get your presentation software ready. LibreOffice Impress is very similar to Microsoft PowerPoint and works well on the Raspberry Pi. If you don't need the other applications in the LibreOffice suite, you can just install Impress and the core components using:

```
sudo apt-get install libreoffice-impress
```

You're going to use Python code to detect gestures via the Skywriter and generate keyboard taps to control Impress in the normal way. The only extra configuration necessary is to activate the functionality that lets you draw on slides with the mouse. This is done by selecting Slide Show>Slide Show Settings and checking the 'Mouse pointer as pen' box.

>STEP-05

Use the code

Type up the code from the listing (right) and save it as **magic_control.py**. You can do this using IDLE or with another text editor of your choice. Start LibreOffice Impress and open your presentation. Then minimise Impress and run your code either through Idle or via the command line:

```
sudo python magic_control.py
```

Now flip back to Impress and get ready to start your talk or presentation.

>STEP-06

Dazzle your audience

Start the slideshow by flicking upwards across the Skywriter. You can navigate through the slides by flicking left to right (forward), or right to left (backwards). To end the presentation, flick down across the Skywriter.

To annotate a slide, you need to activate the drawing function. Do this using a tap/touch: you'll now be controlling the mouse with your hand movements, leaving a trail on the screen just like you're drawing with a pen. This takes practice, so make sure you've spent some time preparing in advance! When you've finished drawing, tap/touch again to disengage the pen function.

We all know that, under the pressure of a live performance, things can go wrong. If you want to quickly disable Skywriter control of the presentation, use a double-tap to quit the Python program.

Magic_Control.py

Language
>PYTHON

DOWNLOAD:
github.com/top-shed/MagicPres

```
#!/usr/bin/env python
import skywriter
import signal
import autopy
import sys

mouse_down = False
#work out how big the screen we're using is.
width, height = autopy.screen.get_size()

@skywriter.move()
def move(x, y, z):
  #print( x, y, z )
  global mouse_down
  if mouse_down: # Only run if we're in drawing mode
      x = (x) * width
      y = (y) * height
      # scale to screen size
      x = int(x)
      y = height - int(y)

      if( y > 799 ):
       y = 799

      autopy.mouse.move(x, y)
      #print( int(x), int(y) )

@skywriter.flick()
def flick(start,finish):
  print('Got a flick!', start, finish)
  if start == "east": # Back through Impress slides
    autopy.key.tap(autopy.key.K_LEFT)
  if start == "west": # Forward through Impress slides
    autopy.key.tap(autopy.key.K_RIGHT)
  if start == "north": # Start slideshow
    autopy.key.tap(autopy.key.K_F5)
  if start == "south": # Quit slideshow
    autopy.key.tap(autopy.key.K_ESCAPE)


@skywriter.double_tap()
def doubletap(position):
  print('Double tap!', position)
  sys.exit() # Emergency stop


@skywriter.tap()
def tap(position):
  global mouse_down
  print('Tap!', position)
  if mouse_down: # Toggle mouse up/dwon
      autopy.mouse.toggle(False)
      mouse_down = False
  else:
      autopy.mouse.toggle(True)
      mouse_down = True

#@skywriter.touch()
#def touch(position):
#  print('Touch!', position)

signal.pause()
```

JAMES SINGLETON

James is a software developer and engineer. He founded Computing 4 Kids Education (C4KE) and Yo Flow online gym inductions.
computing4kids.com
unop.uk

TIME-LAPSE PHOTOGRAPHY

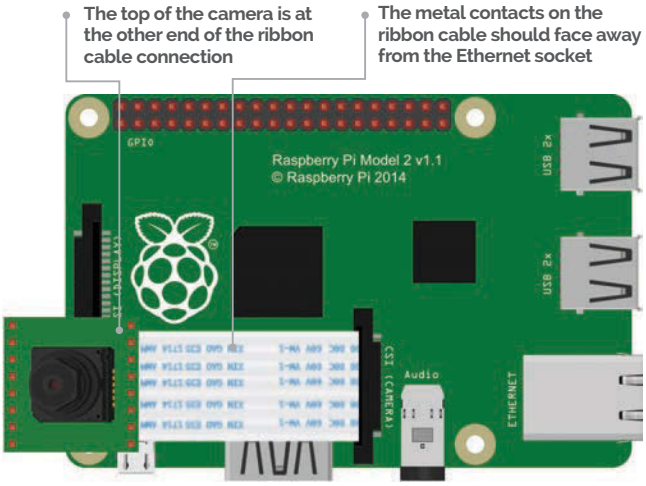
Photo by NASA JSC

You'll Need

- Raspberry Pi Camera Module **raspberrypi.org/products/camera-module**

Make a device to capture photographs at regular intervals. Then turn these images into a video

Time-lapse photography reveals exciting things about the world which you wouldn't otherwise be able see. Things that happen too slowly for us to perceive: bread rising and plants growing; the clouds, sun, moon, and stars crossing the sky; shadows moving across the land. In this tutorial, we'll be making a Raspbian-based device that lets you watch things that are too slow to observe with the naked eye. To do this, we will capture lots of still photographs and combine these frames into a video with FFmpeg/libav, which can then be accessed via a web browser.



>STEP-01

Connect the Camera Module

First, connect the camera module to the Raspberry Pi with the included ribbon cable. Locate the correct socket; it's on the top of the Raspberry Pi circuit board and is the one furthest away from the micro-USB power connector. The socket is handily labelled 'CAMERA' on the newer Raspberry Pi models. Lift up the outside of the socket to release the clamp, then insert the ribbon cable with the metal contacts facing towards the micro-USB power connector. Finally, hold the ribbon cable in position and push the outside of the socket back down to clamp the cable in place.

>STEP-02

Enable and test the camera

Power the Raspberry Pi up. You now have a choice: boot to the command line, open a terminal window, or establish a secure shell (SSH) connection. Enable the camera by running this command from a terminal to launch the Raspberry Pi configuration tool:

```
sudo raspi-config
```

Then select the 'Enable Camera' option. You can test the camera by running the following command:

```
raspistill -o testimage.jpg
```

The red LED on the camera module should light up for 5 seconds and a JPEG image will be saved to the current directory. If the camera is mounted upside down, then you can use the vertical and horizontal flip command-line switches (**-vf** and **-hf**).

>STEP-03

Install and configure software

Install a web server to access your images remotely. Run this command to install Apache:

```
sudo apt-get install apache2
```

Remove the default page to see the contents of the directory:

```
sudo rm /var/www/index.html
```

Visit the IP address of your Pi (e.g. **http://192.168.1.45** – you can find this by using **ifconfig**) and you should see an empty directory listing. If you run the following command and refresh the page, you should see an image file listed. You run this as a superuser so you can write to the directory.

```
sudo raspistill -o /var/www/testimage.jpg
```

Click on the file link and you'll see the image in your browser.

>STEP-04

Capture the images

Set up your scene and check the positioning of the camera.

```
sudo raspistill -w 1920 -h 1080 -o /var/
www/testimageFullHD.jpg
```

The width and height have been changed to capture a smaller image in 16:9 aspect ratio. This makes things easier later. The top and bottom are cropped,

Below Some bread dough ready to prove. Watch it rise in your video. Be careful not to move the bowl or camera during filming

```
pi@raspberrypi ~ $ sudo avconv -i /var/www/frame%04d.jpg -crf 4 -b:v 10M /var/www/video.webm &
[1] 3761
pi@raspberrypi ~ $ avconv version 9.14-6:9.14-1rpi1rpi1, Copyright (c) 2000-2014 the Libav devel
  built on Jul 22 2014 15:08:12 with gcc 4.6 (Debian 4.6.3-14+rpi1)
Input #0, image2, from '/var/www/frame%04d.jpg':
  Duration: 00:00:30.64, start: 0.000000, bitrate: N/A
    Stream #0.0: Video: mjpeg, yuvj420p, 1920x1080, 25 fps, 25 tbr, 25 tbn
[libvpx @ 0x1b72c40] v1.1.0
Output #0, webm, to '/var/www/video.webm':
  Metadata:
    encoder         : Lavf54.20.4
    Stream #0.0: Video: libvpx, yuv420p, 1920x1080, q=-1--1, 10000 kb/s, 1k tbn, 25 tbc
Stream mapping:
  Stream #0:0 -> #0:0 (mjpeg -> libvpx)
Press ctrl-c to stop encoding
frame=  133 fps=  0 q=0.0 size=    6294kB time=5.32 bitrate=9691.9kbits/s
```

so make sure that your subject is in frame. Run this to start the capture:

```
sudo raspistill -w 1920 -h 1080 -t 10800000
-tl 10000 -o /var/www/frame%04d.jpg &
```

This takes a photograph every ten seconds (10,000 milliseconds) for three hours (10,800,000 milliseconds). The ampersand (**&**) at the end runs the process in the background.

Above Shell running the rendering process on the Raspberry Pi. This will take some time, so you may prefer to use a faster machine

MAKE AN ANIMATED GIF

Instead of video, make an animated GIF with ImageMagick. Use smaller images, captured less frequently.

sudo convert /var/www/frame*.jpg /var/www/anim.gif &

>STEP-05

Prepare to make the video

You can render the video on the Raspberry Pi, but it'll be very slow. A better way is to transfer the files to a more powerful computer. In any case, you'll need to install the tools on the rendering machine; for the Pi, enter:

```
sudo apt-get install libav-tools
```

This installs a fork of FFmpeg, but you can also use the original FFmpeg. To copy the images to a remote machine, you can download them from the web server using wget or curl. For example:

```
wget -r -A jpg http://192.168.1.45
```

Or if you don't have wget…

```
curl http://192.168.1.45/frame
[0001-0766].jpg -O
```

Change the IP address and numbers accordingly.

>STEP-06

Make the video

The final step is to make the video. Run this command to start the rendering process:

```
sudo avconv -i /var/www/frame%04d.jpg
-crf 4 -b:v 10M /var/www/video.webm &
```

When this has finished, you'll be able to view the video in your browser. The default frame rate is 25fps. This compresses three hours of frames at ten-second intervals to about forty seconds of video. You can adjust this with the **-framerate** command-line option. The bitrate (**-b**) has been set high, and the Constant Rate Factor (**-crf**) low, to produce a good-quality video.

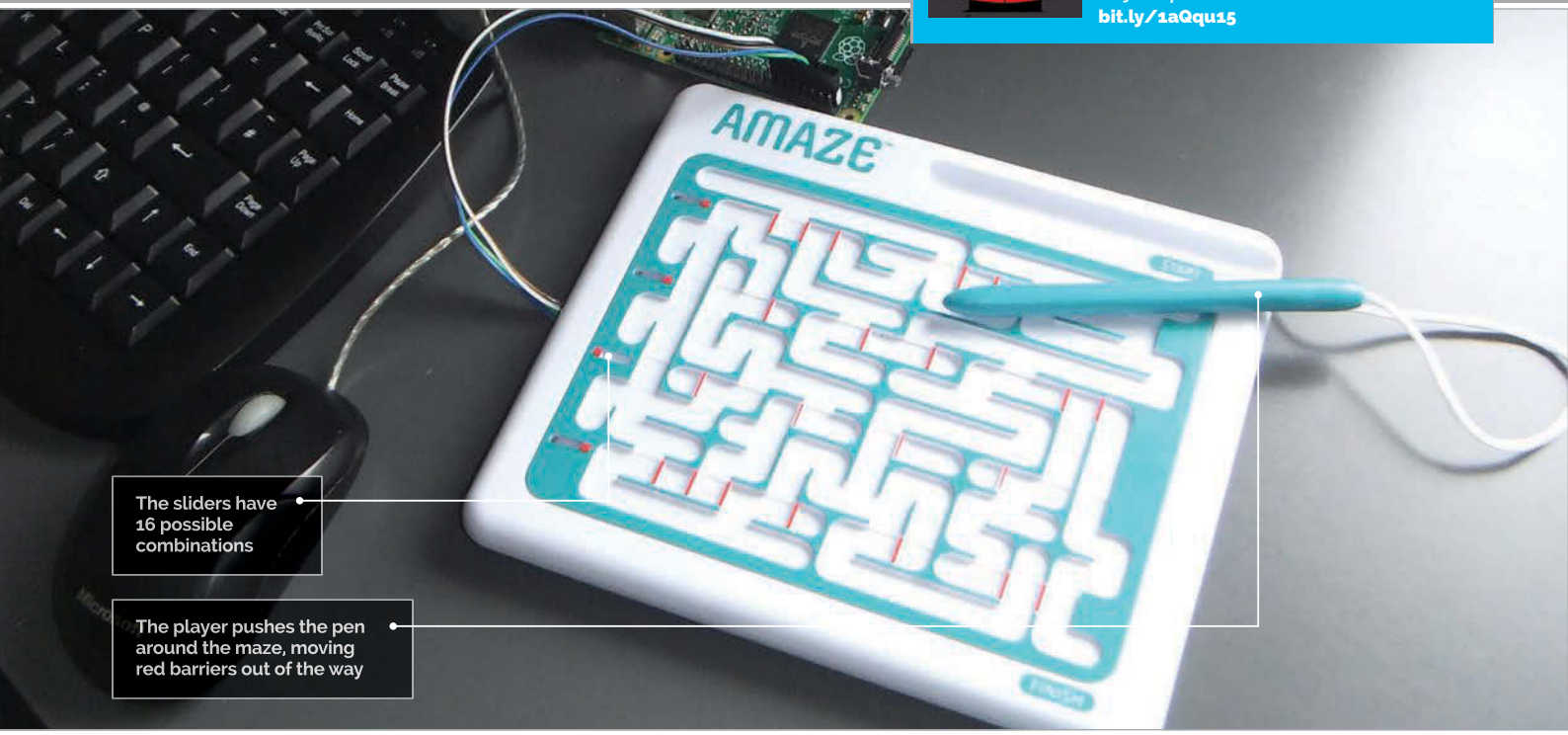
OTHER VIDEO FORMATS

WebM is an open video format that can be displayed directly in most browsers. However, other video formats are available.

MIKE'S PI BAKERY

MIKE COOK

Veteran magazine author from the old days and writer of the Body Build series. Co-author of *Raspberry Pi for Dummies*, *Raspberry Pi Projects*, and *Raspberry Pi Projects for Dummies*.
bit.ly/1aQqu15



You'll Need

- Amaze game by ThinkFun
- 4× SFH3410 light sensors
- 4× pieces of 3- by 4- hole stripboard
- An 8-pin header socket
- 5× 300mm wires with crimped female header sockets
- Insulating tape & hot-melt glue

AMAZE – THE MAZE

Explore the complexity of a moving-barrier maze and let your Raspberry Pi help you navigate it

Here at Mike's Pi Bakery we have always been interested in mazes. The problem tends to be that most mazes are either too open and simple, or too dense and complex. However, recently we came across the Amaze from ThinkFun. This is a maze with a difference: it's quite open, but it's not immediately apparent how to complete it. This is due to four rows of movable barriers on every third line. These barriers can be pushed in the direction that your maze-following pen allows, so the path from start to finish is not initially visible. The game is purely mechanical, but when we got it home we saw that by interfacing it to a Raspberry Pi we could add some fun to the game, both in setup and play. Basically, with four sliders that can be set either left or right, there are 16 possible ways to configure the maze.

The instruction leaflet shows each combination of left and right for an increasingly complicated maze. Furthermore, there is a table telling you which slider you have to push, and in what order, to complete the maze. Of course, even if you got the correct order of pushing, there's no guarantee that you could finish the maze because it matters where in the barrier you pushed it, and they aren't telling you that. What makes this table a bit messy is that you can see all the solutions at once; with a computer interface, not only could you simply select the maze's complexity, but also tailor the sort of help you're given. Add to that the inclusion of sound effects, and the magic of seeing the on-screen graphics reflect the real-life hardware, and you have all the ingredients for a worthy Bakery project.

The project

We needed a way to interface the Raspberry Pi to the game so that it could read the position of each of the sliders; when you move them, the Pi should be able to read the new position and reflect that on the screen. A careful examination of the sliders showed that the round spots at the end covered up or revealed the plastic for the lower half of the case. We thought that a strategically placed phototransistor could detect the light difference between the two states of the slider, especially if a hole was drilled in the lower side. To increase the contrast we placed a piece of black insulating tape on the underside of the slider, because it was a bit translucent. To simplify the electronics circuitry even further, we used a phototransistor attached directly into a digital input and allowed the internal pull-up resistor to act as the load. The only downside to this was that we had to ensure there was sufficient light reaching the phototransistor down a rather narrow hole. If there is insufficient light to register a logic low on the GPIO pin, then a small anglepoise lamp can be aimed at the sliders. The schematic of the sensor interface is shown in **Fig 1** (page 61); we chose the small phototransistor SFH3410 to use here because it was cheap and flat enough to sit in the underside well of the game. For a step-by-step guide to installing the sensors on the Amaze, see the 'Making the interface' box.

The screen display

Now to work on the screen display. We started off with a photograph of the Amaze and, using a photo-editing package, removed all the red partitions and slider indicator with the clone tool. We also removed the cord of the pen and this was then used as the basic background image, stored in an **images** directory at the same level as the main code. Then the program could draw in the red barriers in the appropriate place, depending on the position of the sliders. However, this looked a little jerky going from one position to the other, so we decided to animate the change in position so that it looked like a smooth movement. The results were quite impressive and certainly better than we were expecting. There are a lot of magic numbers involved here for the positions of the red barriers, but these were easily gathered by the program itself by getting it to print out the coordinates of every mouse click. This is commented out in the final code, but you can easily uncomment if you want to see them. The space for the pen holder in the image was used to show user messages, and the 'Start' and 'Finish' labels were used as clickable areas or buttons.

Testing the sensors

The first thing you want to do is to check out the sensors to see if you are getting enough light down through the holes when the slider uncovers them. For this you can use the **Photo_test.py** code on page 63;

MAKING THE INTERFACE

>STEP-01

Drilling the sensor holes

Start off by drilling a hole in the middle of each slider position. Move all the sliders to the right, exposing the base of the box through the slot. Then take a 2mm drill and drill a hole in the middle of the slot; the hole will emerge through the base on the underside. On the reverse, drill a similar-sized blind hole about 1cm away from the through hole. This is to act as a securing hole for the hot-melt glue you will use later.

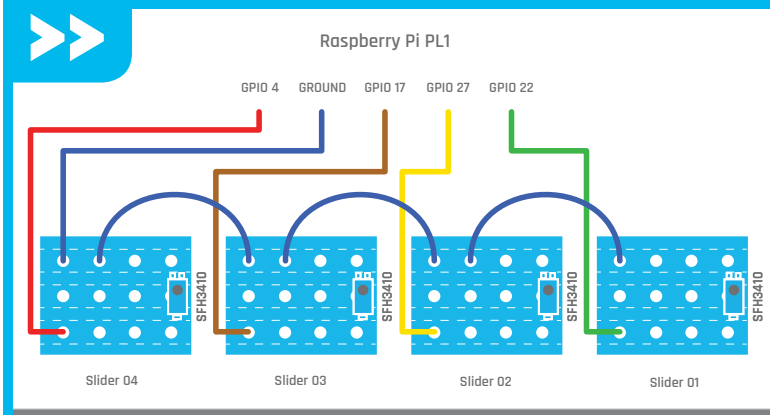
>STEP-02

Fitting light baffles

Dismantle the Amaze by removing the four self-tapping screws, clean out any swarf created by the drilling, and fix a small length of black masking tape inside the back of the slider at the end you have drilled the hole. This reduces the light through a closed hole. You could use acrylic paint here if you like. Reassemble the box, taking care not to force the self-tapping screws back into the holes – if it feels stiff, then back off until you feel it click and then screw in again. You have to be careful not to strip the thread in the plastic.

SURFACE-MOUNT SOLDERING

The only disadvantage with the photo sensor we have chosen is that it's a surface-mount device. We know many people are scared of soldering them, but there's no reason to be worried, providing you have a fine-tipped soldering iron and a pair of tweezers. The way to do this is to fix the stripboard to the bench with a blob of Blu-Tack. Then apply a little solder to one track. Re-melt this solder and offer the sensor up to the solder with the tweezers. Make sure it is aligned correctly and when it is, remove the iron. Then you can solder the other side of the sensor, and finally solder the third unconnected tab for mechanical stability. We have made a video of this process and you can see it in the Pi Bakery video channel at vimeo.com/138562025.



>STEP-03

Wiring up the sensors

Make up four sensor boards using small pieces of stripboard, and wire them up as shown in the diagram. We used a single-width 8-pin header socket to connect these wires to the Raspberry Pi's GPIO connector PL1. The first three pin connections are empty, and the GPIO pins we used followed on from that. We marked the end of this connector with typing correction fluid to make sure we always connected it the correct way round to the Pi's port. You could use a blob of white paint if you haven't got any corrector fluid.

>STEP-04

Fixing the sensors

Fix the boards to the back of the Amaze case with hot-melt glue. Make sure that you can see the black spot of the sensor through the hole from the other side before letting the glue set. Tack the board on at first to get the position correct, and then fill in the remaining spaces with glue. Make sure you do this in stages so as not to melt and disturb the initial positioning spot of glue. When set, cover all the sensor boards with a strip of black insulation tape to ensure a light, tight fitting for the back.

this simply initialises the four GPIO pins as inputs, reads them at one-second intervals and prints out the value. Note here that a logic 1 indicates that there's no light or not enough reaching the sensor, and a logic 0 shows that the sensor is detecting enough light to pull the pin low. For the test, expose all the sensors by moving all the sliders to the right, adjust the lighting so that all the sensors read zero, and then push the sliders to the left, making sure that each one goes to a logic 1.

Using the program

With the sensors in place and the light adjusted, it's time to run the **Amaze.py** code listing. It's a Pygame program, so clicking the close box or pressing **ESC** will quit the program. There's a startup message printed to the Python console, but after that all communication is through the graphic display. The first thing to do is to select the maze number; these go from 1 to 16 and the bigger the number, the more complex the maze is. When you have selected the one you want, click on the Start label; the display will then move the sliders into the start position and you have to physically push the sliders to match this. When you have done this, you will be asked to set a help level; there are three of them. First, there is no help at all. Next, there is the validation of a slider move; that means if you move a slider, then you will be told if this was the correct slider to move. The final level of help tells you which slider is the next one to move; however, there is no indication of where in the slider you must push it, so it is still possible to trap yourself even if you move the correct slider. When you have finished or want to give up, click the on-screen Finish label and the whole process will start again. Let's look at the code and see how this magic is worked.

How the code works

After the normal Pygame initialisation stuff there are four lists defined, one for each slider, containing a tuple with the x and y coordinate of the slide indicator and the start of each of the barriers. These lists are then combined into a list of lists. The same sort of thing is done with the maze solutions. For each initial starting slider configuration of a maze, there is a list of sliders to push, and this list is combined into a list of lists. Note here that while the sliders are numbered for the user as being 1 to 4, in the code these will be given the numbers 0 to 3, so when indicating the next slider to move a 1 must be added to the number in the list. The same goes for maze starting numbers, 1 to 16. This time the problem is resolved by putting a blank list as the first item in the list of lists. Next, the sounds are loaded in from a **sounds** directory.

The main function is mainly one infinite **`while`** loop; the first part calls the functions that set up the game, and the final **`while`** loop runs until the finish is indicated. The program only responds to a slider move once it's running, and this is done by the **`updateSensors`** function. This function checks to see if any of the sensors indicate a different slider condition than the program has displayed; if so, then that slider is updated with the **`toggleSlider`** function. It first looks at the old slider state to see what direction it needs to move to. Then it performs the animation of moving the sliders; after that, it determines and displays any feedback messages. This function is also responsible for playing most of the feedback sounds. The **`checkForEvent`** function handles all the

mouse clicks; there is a section commented out that will 'push' the sliders from the keyboard – useful for testing, but not for running the game.

Customisation

So, what can you do to add your own twist to this project? Well, the observant will have noticed that the clickable areas to set the maze number and help levels remain active at all times; you could make the program more robust by only allowing those clicks to affect the numbers at the correct time. You can do this simply by using a global Boolean variable in the click condition, and only setting those variables when the appropriate function is being executed. Also, when clicking the Start and Finish labels, it would be good to invert or highlight those areas to give some feedback. You might want to add white LEDs above each sensor so that the shadow of your hand doesn't accidentally trigger the slide sensors. However, one big improvement you could make is to give help as to which barrier on the slider to push, but that would require quite a bit more work analysing the maze. Whatever you do, have fun.

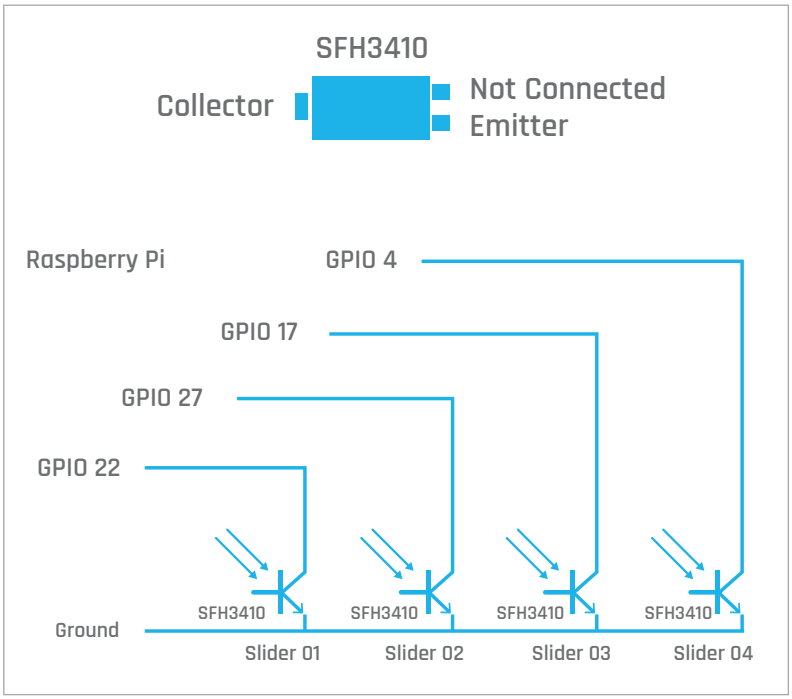


Fig 1 The schematic of the slide position sensors

Amaze.py

```
# Amaze - maze game
# By Mike Cook - September 2015

import pygame, time, os
import wiringpi2 as io

pygame.init() # initialise graphics interface
pygame.mixer.quit()
pygame.mixer.init(
frequency=22050, size=-16, channels=2, buffer=512)

os.environ['SDL_VIDEO_WINDOW_POS'] = 'center'
pygame.display.set_caption("Amaze - Maze helper")
pygame.event.set_allowed(None)
pygame.event.set_allowed([pygame.MOUSEBUTTONDOWN,pygame.
KEYDOWN,pygame.QUIT])
screen = pygame.display.set_mode([800,665],0,32)
textHeight = 36
font = pygame.font.Font(None, textHeight)
# define the points to draw the barriers
slider1 = [(87,173),(172,159),(172,187),(210,159),(210,186),
(248,157),(248,186),
          (401,157),(401,185), (440,157),(440,185),(516,157),
(516,185)]
slider2 = [(84, 285),(209, 270),(209, 300),(325, 269),
(325, 299),(403, 269),
          (403, 298),(558, 269),(558, 299),(596, 269),
(596, 298)]
slider3 = [(81, 402),(207, 387),(207, 417),(327, 385),
          (327, 416),(445, 384),(445, 415),(563, 384),
(563, 414)]
slider4 = [(78, 522),(166, 507),(166, 538),(206, 506),
(206, 537),(246, 505),
          (246, 538),(327, 505),(327, 538),(448, 505),
          (448, 536),(568, 504),(568, 535),(607, 503),
(607, 535)]
slider = [slider1, slider2, slider3, slider4]
sliderShift = [38,38,40,40] # amount to shift slider x position
for each line
sliderState = [True,True,True,True]
# True = left, False = right
mazeLook = [5,9,4,11,6,10,1,12,7,13,3,
14,8,15,2,16]
textHeight = 36
font = pygame.font.Font(None,
textHeight)
# solutions
m1  =[3]
m2  =[3]
m3  =[1,0,3]
m4  =[1,0,0,3]
m5  =[2,1,0,0,3]
m6  =[1,2,1,0,0,3]
m7  =[0,2,1,0,0,3]
m8  =[1,0,2,1,0,0,3]
m9  =[0,3,0,2,1,0,0,3]
m10 =[1,0,3,0,2,1,0,0,3]
m11 =[2,0,3,0,2,1,0,0,3]
m12 =[2,1,0,3,0,2,1,0,0,3]
m13 =[1,0,1,0,3,0,2,1,0,0,3]
m14 =[2,1,0,1,0,3,0,2,1,0,0,3]
m15 =[1,1,0,1,0,3,0,2,1,0,0,3]
m16 =[2,1,1,0,1,0,3,0,2,1,0,0,3]
sol =[ [],m1,m2,m3,m4,m5,m6,m7,m8,m9,m10,m11,m12,m13,m14,m15,
m16 ]
progress = 0


pinList = [22,27,17,4] # GPIO pins for sensor switches
background =  pygame.image.load(
"images/noSlide.jpg").convert_alpha()
restart = False
chosen = False # for options
mazeIDnumber = 1
helpLevelNumber = 2
helpLevel = ["No help", "Validate slide",
"Show next slide","Start maze run now"]
soundEffects = ["slide","ping","wrong","finish","thanks","help"]
mazeSound = [ pygame.mixer.Sound(
"sounds/"+soundEffects[sound]+".wav")
```

Language
>PYTHON 2.7

DOWNLOAD:
github.com/Grumpy-Mike/Mikes-Pi-Bakery/tree/master

```
                for sound in range(0,6)]

def main():
    global moved, restart, progress
    initGPIO()
    print"Amaze - click on the start button"
    while True:
        restart = False
        setMaze(choose())
        waitForSet() # player puts maze into that position
        progress = 0 # number of slides
        helpLevelRequired()
        if helpLevelNumber == 2:
            drawWords("First slider to move is"
+str(sol[mazeIDnumber][progress]+1),366, 37)
            pygame.display.update()
        while not restart:
          checkForEvent()
          updateSensors()

def initGPIO():
    try :
        io.wiringPiSetupGpio()
    except :
        print"start IDLE with 'gksudo idle' from command line"
        os._exit(1)

    for pin in range (0,4):
        io.pinMode(pinList[pin],0)
        io.pullUpDnControl(pinList[pin],2) # input enable pull up

def showPicture(): # draws maze and sliders
    screen.blit(background,[0,0])
    lineCol = (230,20,0)
    for s in range(0,4):
        pygame.draw.circle(screen,lineCol,slider[s][0],4,0)
        points = len(slider[s])-1
        for block in range(1,points,2):
            pygame.draw.line(screen,lineCol,
slider[s][block],slider[s][block+1],4)
    pygame.display.update()

def waitForSet(): # hold until put into matching position

    drawWords("Set your Maze like this",366, 37)
    pygame.display.update()
    while checkSetup():
        checkForEvent()
    showPicture()
    drawWords("Thank you",366, 37)
    pygame.display.update()
    mazeSound[4].play()
    time.sleep(3.0)

def checkSetup(): # see if sensors are registering correctly
    done = True
    for i in range(0,4):
        if sliderState[i] != bool(io.digitalRead(pinList[i])):
            done = False
    return not done

def updateSensors(): # see if the sliders have changed
    for i in range(0,4):
        sensor = bool(io.digitalRead(pinList[i]))
        if sliderState[i] != sensor:
            toggleSlider(i, True)
            time.sleep(2.0)
    return

def helpLevelRequired():
    global helpLevelNumber, chosen
    drawWords(helpLevel[helpLevelNumber],366, 37)
    drawWords("->",366,65)
    drawWords("Change help level ",398,65)
    pygame.display.update()
    mazeSound[5].play()
    current = helpLevelNumber
    chosen = False
    while not chosen:
        checkForEvent()
        if current != helpLevelNumber :
            current = helpLevelNumber
            showPicture()
            drawWords(helpLevel[helpLevelNumber],366, 37)
            drawWords("->",366,65)
            pygame.display.update()
    drawWords(helpLevel[3],366, 37)
    pygame.display.update()
    time.sleep(2)
    showPicture()


def choose():
    global chosen, mazeIDnumber
    chosen = False
    mazeIDnumber = 1
    currentMaze = mazeIDnumber
    showPicture()
    drawWords("Choose Maze",366, 37)
    drawWords("<-",536,65)
    drawWords("->",576,65)
    drawWords("01",556,37)
    pygame.display.update()
    while not chosen :
        checkForEvent()
        if mazeIDnumber != currentMaze :
            currentMaze = mazeIDnumber
            if currentMaze > 9 :
              ns = str(currentMaze)
            else :
              ns = "0"+str(currentMaze)
            drawWords(ns,556,37)
            pygame.display.update()
    return mazeIDnumber


def drawWords(words,x,y) :
        textSurface = pygame.Surface((len(words)*12,textHeight))
        textRect = textSurface.get_rect()
        textRect.left = x
        textRect.top = y
        pygame.draw.rect(screen,(81,133,133), (
x,y,len(words)*12,textHeight-10), 0)
        textSurface = font.render(
words, True, (180,180,180), (81,133,133))
        screen.blit(textSurface, textRect)

def toggleSlider(s,showHelp): # move sliders to the other side
    global sliderState, progress
    if showHelp :
        mazeSound[0].play()
    if sliderState[s] :
        shift = 2
    else :
        shift = -2
    sliderState[s] = not sliderState[s]
    moves = 0
    while moves != sliderShift[s] :
        moves += abs(shift)
        updateSlider(s,shift)
        showPicture()
    if showHelp and helpLevelNumber !=0 :
        time.sleep(0.8) # allow slide sound to finish
        if progress < len(sol[mazeIDnumber]):
            if s != sol[mazeIDnumber][progress] :
                drawWords("Wrong - out of sequence",366, 37)
```

```
            pygame.display.update()
            mazeSound[2].play()
            if helpLevelNumber == 2:
               time.sleep(3.0)
               drawWords("The next slider move should be "
+str(sol[mazeIDnumber][progress]+1),366, 37)
               pygame.display.update()
         else :
            drawWords("OK",366, 37)
            pygame.display.update()
            mazeSound[1].play()
            progress += 1
            if progress < len(sol[mazeIDnumber]) :
               if helpLevelNumber == 2:
                  time.sleep(1.2)
                  drawWords("OK - next slider is "
+str(sol[mazeIDnumber][progress]+1),366, 37)
                  pygame.display.update()
            else:
               time.sleep(1.5)
               mazeSound[3].play()
               drawWords("Click on Finish when done",366, 37)
               pygame.display.update()
      else:
         drawWords("Move was not required",366, 37)
         pygame.display.update()

def updateSlider(s,inc):
   global slider
   points = len(slider[s])
   for i in range(0,points):
      slider[s][i]= (slider[s][i][0] + inc,slider[s][i][1])

def mazeNumber(): # set the maze to an ID number
   mazeNum = 0
   for i in range(0,4):
      mazeNum = mazeNum << 1
      if sliderState[i] :
         mazeNum = mazeNum | 1
   return mazeLook[mazeNum]

def setMaze(n):
   i = 0
   while mazeLook[i] != n:
      i += 1
   # i has the bit pattern of the slider states
   mask = 0x8
   for n in range(0,4):
      if (mask & i) == 0 and sliderState[n] == True:
         toggleSlider(n, False)
      elif (mask & i) != 0 and sliderState[n] == False:
         toggleSlider(n, False)
      mask = mask >> 1

def terminate(): # close down the program
    print "Closing down please wait"
    pygame.mixer.quit()
    pygame.quit() # close Pygame
    os._exit(1)

def checkForEvent(): # look at keys
    global restart, chosen, mazeIDnumber, helpLevelNumber
    event = pygame.event.poll()
    if event.type == pygame.MOUSEBUTTONDOWN:
       point = pygame.mouse.get_pos()
       # print point
       # print out position of click for development
       if (point[0] > 366 and point[0] <520) and (
point[1] > 37 and point[1] < 63):
          chosen = True
       if (point[0] > 630 and point[0] <691) and (
point[1] > 76 and point[1] < 94):
          chosen = True
       if (point[0] > 535 and point[0] <557) and (
point[1] > 67 and point[1] < 91):
          mazeIDnumber -= 1
          if mazeIDnumber < 1:
             mazeIDnumber = 16
       if (point[0] > 576 and point[0] <597) and (
point[1] > 66 and point[1] < 90):
          mazeIDnumber += 1
          if mazeIDnumber > 16:
             mazeIDnumber = 1
       if (point[0] > 365 and point[0] <385) and (
point[1] > 66 and point[1] < 91):
          helpLevelNumber +=1
          if helpLevelNumber >2:
             helpLevelNumber = 0
       if (point[0] > 661 and point[0] <725) and (
point[1] > 607 and point[1] < 623):
          restart = True

    if event.type == pygame.QUIT :
        terminate()
    if event.type == pygame.KEYDOWN :
       if event.key == pygame.K_ESCAPE :
          terminate()
       if event.key == pygame.K_RETURN :
          restart = True
       '''
       if event.key == pygame.K_1 :
          toggleSlider(0, True)
       if event.key == pygame.K_2 :
          toggleSlider(1, True)
       if event.key == pygame.K_3 :
          toggleSlider(2, True)
       if event.key == pygame.K_4 :
          toggleSlider(3, True)
       '''
# Main program logic:
if __name__ == '__main__':
    main()
```

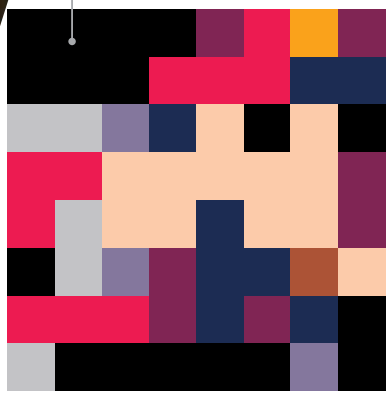
Photo_test.py

```
# photo transistor input test
# for the Amaze game - Mike Cook September 2015
import time, os
import wiringpi2 as io

pinList = [22,27,17,4] # GPIO pins for sensor switches

def main():
   initGPIO()
   print"Amaze sensor switch test"
   while True:
      for i in range(0,4):
         print io.digitalRead(pinList[i])," ",
      print" "
      time.sleep(1)

def initGPIO():
   try :
      io.wiringPiSetupGpio()
   except :
      print"start IDLE with 'gksudo idle' from command line"
      os._exit(1)
   for pin in range (0,4):
      io.pinMode(pinList[pin],0)
      io.pullUpDnControl(pinList[pin],2) # input enable pull up

# Main program logic:
if __name__ == '__main__':
    main()
```

ROB ZWETSLOOT

Tech writer, avid coder, and Raspberry Pi enthusiast with a history of building many things with Raspberry Pi
twitter.com/RobThez

The 8×8 display on the Sense HAT may be simple, but it has a wide range of colours and is fully programmable



Create simple images or more advanced pixel art to represent your favourite thing

You'll Need

- Raspbian **raspberrypi.org/downloads**
- Sense HAT **raspberrypi.org/products/sense-hat**
- Pixel art

PIXEL ART ON SENSE HAT

Make use of Python and the Sense HAT's LEDs to read and display 8×8 pixel art

We know what you're thinking: 8×8 pixel art doesn't seem like a whole lot of pixels. And if you were thinking that, you'd be right. That's 64 pixels to try to convey something – simple for low-complexity objects, but for proper pixel art it seems like a tall order. If you think back to the early days of gaming, though, there are plenty of examples of sprites that didn't use many pixels at all and looked… passable. The main limitations in those days were the number of colours you could have on screen at once, and with the Sense HAT we have the power to use a full RGB spectrum, utilising every pixel to make something a bit better than on the old Atari 2600. Grab a sprite, and let's go!

INSPIRATIONAL CODE

This tutorial was inspired by reader Mike Legg on Twitter, who created code to cycle through all 100 characters on Johan Vinet's sheet. View the Python code for this here: **bit.ly/1GOVoBp**

>STEP-01

Get the Sense HAT ready

If you've upgraded to the newer version of Raspbian, Jessie, all you need to do is switch off the Raspberry Pi, remove the power cable, and then plug the Sense HAT on top of the Pi. Turn it back on and it's ready to use. If you haven't made the update, you'll need to install a few extra libraries first. Open the terminal and run the following commands:

```
sudo apt-get install sense-hat
sudo pip-3.2 install pillow
```

…and then reboot the Raspberry Pi (which you can also do in the terminal with **sudo reboot** if you wish!)

>STEP-02

Find some art

There's a few ways you can do this: using Google or searching on something like DeviantArt with the right keywords ('8×8 pixel art' is a good start) and you should be able to find a suitable sprite for showing off on your Sense HAT. You could also experiment with a square picture (from something like Instagram) to see what the code will spit out, or you can draw your

own pixel art in something like Swanky Paint. For testing this tutorial, we grabbed one of the 100 characters from a sprite sheet created by Johan Vinet (**twitter.com/johanvinet**), which can be found here: **bit.ly/1jxaJ3m**.

>STEP-03

Prepare the art

The Sense HAT can only display 64 pixels, and our code assumes the image you have is going to be exactly that. On a normal computer, use an image editor such as the free software GIMP to prepare the image. Remove any borders (you may need to zoom right in to make sure) by cropping the image and save it to a file name you'll remember with the extension .png or .gif. This is especially important if the 8×8 art you have isn't actually saved as an 8×8 image; for instance, if each 'pixel' in the art is six pixels wide, like in our code.

>STEP-04

Study the pixel art

As mentioned above, your pixel art may not be actually saved by the pixel. In the code we've created, it takes this into account. The easiest way to check the width of the pixels is to look at the resolution of the image. In our case, it's 48×48, which means that each 'pixel' is six real pixels wide (and high, because it's a square). You can check this in GIMP by using a square pixel brush and increasing the size until it's the same as the pixels in your image. Alternatively, once you've cropped the image, you can scale the image to be 8×8; this may not have the desired results, though, as it tries to squash everything together.

>STEP-05

Load it all on the Raspberry Pi

Put the image on the Raspberry Pi and open it in the image viewer to make sure it transferred fine. We did this by plugging the SD card directly into the computer we prepared the image on and copied it to a directory called 'SenseHAT' in the home folder, but you can always put it on a USB stick and copy it over from within the Pi or upload it to your cloud storage and download it to the Pi.

>STEP-06

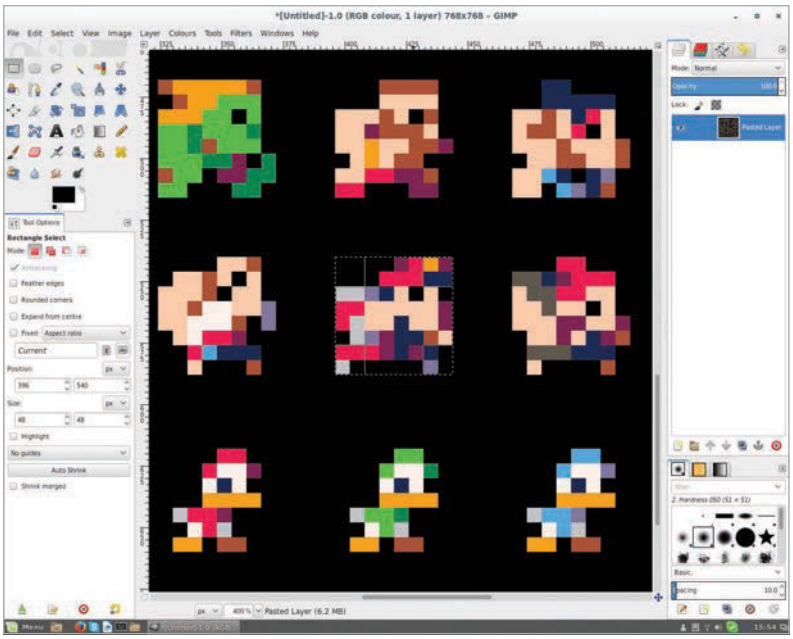
Write the code!

Follow along to the code listing and tweak it to suit your own pixel art, whether this means you need to change the location of the file in the code (our code assumes you've put it in a folder called 'SenseHAT' in Raspbian's home directory) or if you need to change the width of each pixel to 1 or higher. Press F5 to run the code, and it will then display the image on the Sense HAT for three seconds before turning itself off.

Pixelart.py

Language >PYTHON

```
from sense_hat import SenseHat
import time
from PIL import Image
import os

# Open image file
image_file = os.path.join(
os.sep,"/home","pi","SenseHAT","pixelart.png")
img = Image.open(image_file)

# Generate rgb values for image pixels
rgb_img = img.convert('RGB')
image_pixels = list(rgb_img.getdata())

# Get the 64 pixels you need
pixel_width = 6
image_width = pixel_width*8
sense_pixels = []
start_pixel = 0
while start_pixel < (image_width*64):
    sense_pixels.extend(image_pixels[start_pixel:(
start_pixel+image_width):pixel_width])
    start_pixel += (image_width*pixel_width)

# Display the image
sense = SenseHat()
sense.set_rotation(r=180)
sense.set_pixels(sense_pixels)
time.sleep (3)

sense.clear()
```



Above Rob is very vain, and (carefully) cut out an image of his alter-ego M. Bison to display on the Sense HAT, taken from Johan's sprite list

SAM AARON

Sam is the creator of Sonic Pi. By day he's a Research Associate at the University of Cambridge Computer Laboratory; by night he writes code for people to dance to.
sonic-pi.net

Sonic Pi PART 4

ACID BASS

You'll Need

- Raspberry Pi running Raspbian
- Sonic Pi v2.6+
- Speakers or headphones with a 3.5mm jack
- Update Sonic Pi: `sudo apt-get update && sudo apt-get install sonic-pi`

In the fourth part of his series, **Sam Aaron** turns your Raspberry Pi into a TB-303, the infamous acid house bass sound...

It's impossible to look through the history of electronic dance music without seeing the enormous impact of the tiny Roland TB-303 synthesiser. It's the secret sauce behind the original acid bass sound. Those classic squealing and squelching TB-303 bass riffs can be heard from the early Chicago House scene through to more recent electronic artists such as Plastikman, Squarepusher, and Aphex Twin.

Interestingly, Roland never intended for the TB-303 to be used in dance music. It was originally created as a practice aid for guitarists. Roland imagined that people would program the TB-303 to play basslines to jam along to. Unfortunately, there were a number of problems: it was a little fiddly to program, didn't sound particularly good as a bass-guitar replacement, and was pretty expensive to buy. Opting to cut its losses, Roland stopped making the TB-303 after 10,000 units were sold. After a number of years sitting on guitarists' shelves, many ended in the windows of second-hand shops. These discarded TB-303s were waiting to be discovered by a new generation, which started experimenting and using them to create new crazy sounds. Acid house was born.

While getting your hands on an original TB-303 isn't so easy, you'll be pleased to know that you can turn your Raspberry Pi into one using the power of Sonic Pi. Just put this code into an empty buffer and hit Run:

```
use_synth :tb303
play :e1
```

Instant acid bass! Let's play around...

Squelch that bass

First, let's build a live arpeggiator to make things fun. In the last tutorial, we looked at how riffs can just be a ring of notes that we tick through one after another, repeating when we get to the end. Let's create a live loop that does exactly that:

```
1. use_synth :tb303
2. live_loop :squelch do
3.   n = (ring :e1, :e2, :e3).tick
4.   play n, release: 0.125, cutoff: 100,
   res: 0.8, wave: 0
5.   sleep 0.125
6. end
```

Take a look at each line...

1. On the first line, we set the default synth to be **`tb303`** with the **`use_synth`** function.

2. On line two, we create a live loop called **`:squelch`**, which will just loop round and round.

3. Line three is where we create our riff – a ring of notes (E in octaves 1, 2, and 3), which we simply tick through with **`.tick`**. We define **`n`** to represent the current note in the riff. The equals sign just means to assign the value on the right to the name on the left. This will be different every time round the loop. The first time round, **`n`** will be set to **`:e1`**. The second time round, it will be **`:e2`**, followed by **`:e3`**, and then back to **`:e1`**, cycling round forever.

4. Line four is where we actually trigger our **:tb303** synth. We're passing a few interesting opts here: **release:**, **cutoff:**, **res:**, and **wave:**, which we will discuss below.

5. Line five is our **sleep** – we're asking the live loop to loop round every **0.125** seconds, which works out at eight times a second at the default BPM of 60.

6. Line six is the **end** to the live loop. This just tells Sonic Pi where the end of the live loop is.

Whilst you're still figuring out what's going on, type in the code above and hit the Run button. You should hear the **:tb303** kick into action. Now, this is where the action is: let's start live coding.

Whilst the loop is still live, change the **cutoff:** opt to **110**. Now hit the Run button again. You should hear the sound become a little harsher and more squelchy. Dial in **120** and hit Run. Now **130**. Listen how higher cutoff values make it sound more piercing and intense. Finally, drop it down to **80** when you feel like a rest. Then repeat as many times as you want. Don't worry, I'll still be here…

Another opt worth playing with is **res:**. This controls the level of resonance of the filter. A high resonance is characteristic of acid bass sounds. We currently have our **res:** set to **0.8**. Try cranking it up to **0.85**, then **0.9**, and finally **0.95**. You might find that a cutoff such as **110** or higher will make the differences easier to hear. Now, go crazy and dial in **0.999** for some insane sounds. At a **res:** this high, you're hearing the cutoff filter resonate so much that it starts to make sounds of its own!

Finally, for a big impact on the timbre, try changing the **wave:** opt to **1**. This is the choice of source oscillator. The default is **0**, which is a sawtooth wave. **1** is a pulse wave and **2** is a triangle wave.

Of course, try different riffs by changing the notes in the ring or even picking notes from scales or chords. Have fun with your first acid bass synth.

Deconstructing the TB-303

The design of the original TB-303 is actually pretty simple. There are only four core parts. First is the oscillator wave – the raw ingredients of the sound. For instance, this could be a square wave. Next, there's the oscillator's amplitude envelope, which controls the amp of the square wave through time. These are accessed in Sonic Pi by the **attack:**, **decay:**, **sustain:**, and **release:** opts, along with their level counterparts. For more information, read Section 2.4, 'Duration with Envelopes', of the built-in tutorial. We then pass our enveloped square wave through a resonant low-pass filter. This chops off the higher frequencies, as well as having that nice resonance effect. Now this is where the fun starts. The cutoff value of this filter is also controlled by its own envelope! This means we have amazing control over the timbre of the sound by playing with both of these envelopes. Let's take a look:

```
use_synth :tb303
with_fx :reverb, room: 1 do
  live_loop :space_scanner do
    play :e1, cutoff: 100, release: 7,
attack: 1, cutoff_attack: 4,
cutoff_release: 4
    sleep 8
  end
end
```

For each standard envelope opt, there's a **cutoff_** equivalent opt in the **:tb303** synth. So, to change the cutoff attack time, we can use the **cutoff_attack:** opt. Copy the code above into an empty buffer and hit Run. You'll hear a crazy sound warble in and out. Now start to play around with it. Try changing the **cutoff_attack:** time to **1** and then **0.5**. Now try **8**.

Notice that I've passed everything through a **:reverb** FX for extra atmosphere – try other FX to see what works!

Bringing it all together

Finally, here's a piece I composed using the ideas in this tutorial. Copy it into an empty buffer, listen for a while, and then start live-coding your own changes. See what crazy sounds you can make with it! See you next time…

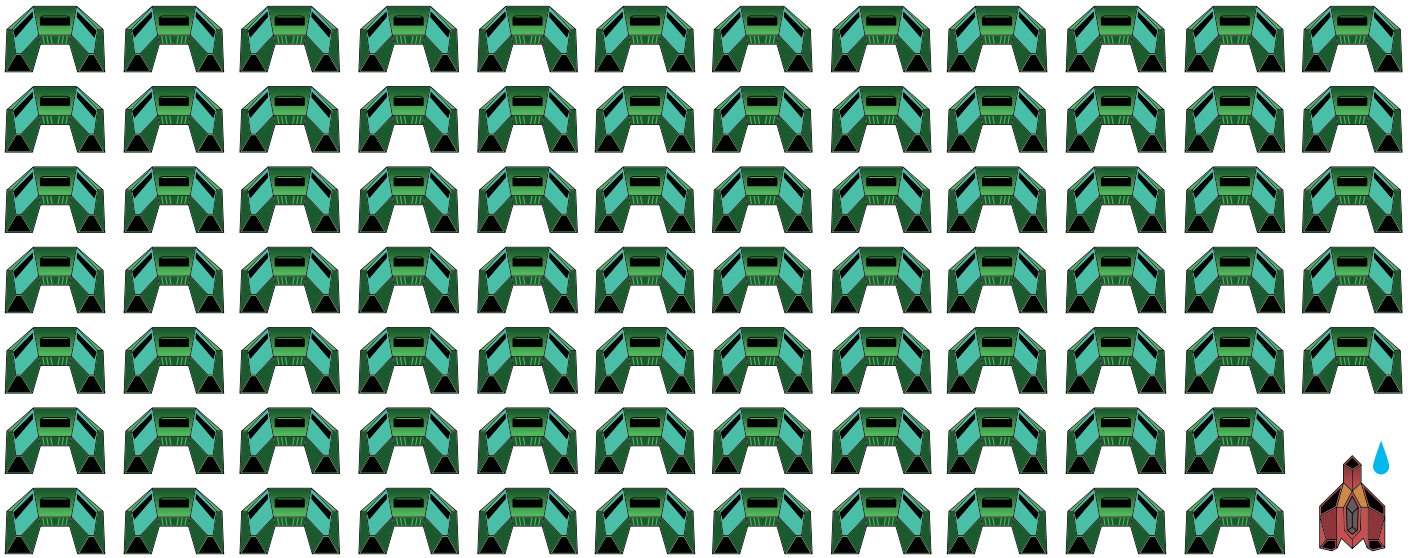
```
use_synth :tb303
use_debug false

with_fx :reverb, room: 0.8 do
  live_loop :space_scanner do
    with_fx :slicer, phase: 0.25, amp: 1.5
do
      co = (line 70, 130, steps: 8).tick
      play :e1, cutoff: co, release: 7,
attack: 1, cutoff_attack: 4,
cutoff_release: 4
      sleep 8
    end
  end

  live_loop :squelch do
    use_random_seed 3000
    16.times do
      n = (ring :e1, :e2, :e3).tick
      play n, release: 0.125, cutoff:
rrand(70, 130), res: 0.9, wave: 1,
amp: 0.8
      sleep 0.125
    end
  end
end
```

PART 9

MAKE GAMES WITH PYTHON

SEAN M TRACEY

Sean is a technologist living in the South West of England. He spends most of his time making silly things with technology.
sean.mtracey.org



THE ALIENS ARE TRYING TO KILL ME [PART 01]

In the penultimate part of this series, we make the first half of a game, putting to use everything we've learned in the first 8 instalments...

Hello again – we're almost at the end! This is the penultimate part of this series on making games with Pygame. We've got a lot to do, so we'll jump straight in to the action. Over this part and the final part, we're going to use everything we've learned so far to make a space-shooter game. Using our mouse, we're going to control a small but feisty spaceship that fends off wave after wave of merciless alien hordes, by blasting them with a lethal green laser-ray (patent-pending). In the final part of this series we'll learn how to make levels with organised structures, instead of the random placement of enemies that we have in this version of our game.

Below The start screen for our final game

We'll also add a game-over screen, some UI elements like health and number of bullets, and we'll add some shields to our space vessel too, because who doesn't like shields?

Since we're not learning anything new this time, we don't need to explore any abstract game or programming concepts before we can make something; we're just going to walk through the code and figure out what we've done and why we've done it that way. So let's look at the code first – specifically, the structure of our code. You may notice that the code for our game is not in one big file as it has been for most of our previous games. Instead, it's split across three separate files: one for our main game logic (we've called it **aliens.py**), one that

Above A screenshot of the game that we'll make in this half of the tutorial

contains the code for our spaceships (**ships.py**), and one file that contains all of the information about our lasers and bullets (**projectiles.py**).

aliens.py is where we will run our game from. It is responsible for handling how we react to user interactions and game events, such as moving and firing the ship, creating new enemies, and triggering sounds. **ships.py** and **projectiles.py** will be imported by **aliens.py**, and will be used to create our own spaceship, the enemy space ships, and the projectiles of both of these types of ship.

Aliens.py

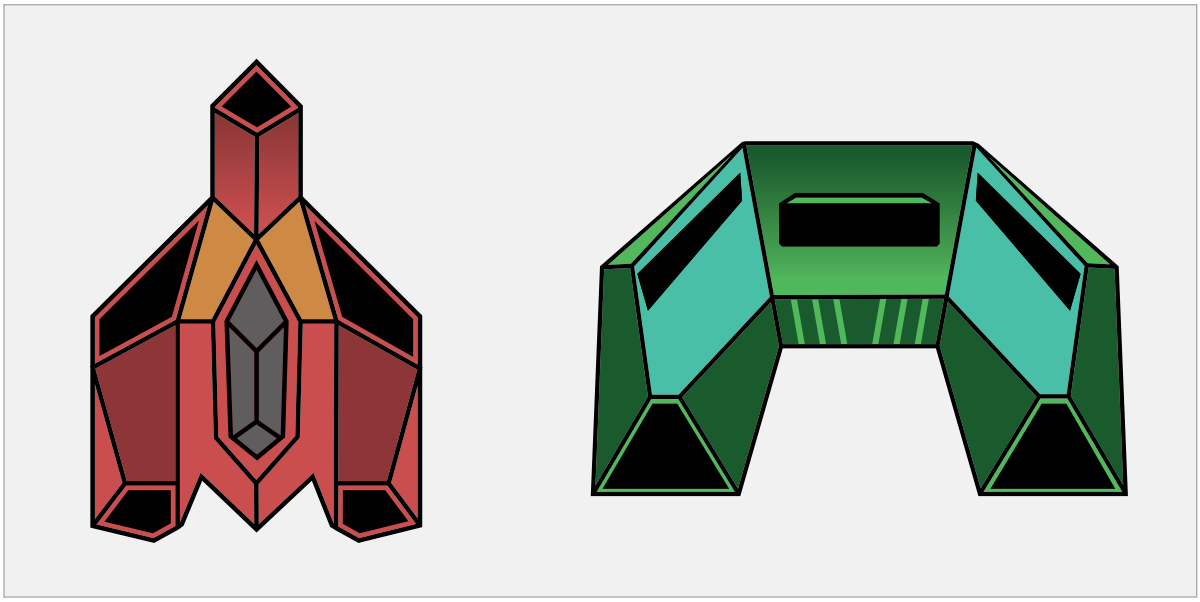
Let's break down the structure of **aliens.py** first. This is where everything in the game will start from, so it makes sense that we should too. In what must seem like the bleeding obvious to you now, on lines 1–5 we have our import statements. Here, we're importing all of the standard Python and Pygame modules that we'll need to make our game do its thing. We also import our own file, **ships.py**, which sits in the same folder as our **aliens.py** folder, with `import ship`.

On lines 7–39, we have all of the global variables that we'll use to keep track of the various objects and events that occur in our game. These are 'global' variables because they don't fall within the scope of any function in our program; this means that any function in our game can read and change the variables as they like. In a lot of circumstances this is frowned upon, but for games it's perfect. Not every variable we need to make this game is declared here; there are quite a few more in our ships and projectile classes that we'll get to shortly. Remember, using classes is a great way to keep properties that are relevant to the thing we're using all wrapped up nicely together.

In previous tutorials, we've always used a main loop to update our game state and draw the items and objects about the place as we've needed. We're breaking that habit this time. Instead of having the main loop be the sole place where game stuff happens, we're going to create two extra functions and split up the work between them. These two functions are `updateGame()` and `drawGame()`, and their purposes are pretty simple. `updateGame()` (lines 41–75) will always be called before `drawGame()` (lines 77–84); it will update everything that has changed since the last loop through our main function. It will also calculate the new positions and statuses of all of our game objects if they've changed, and then update those properties. Straight after that, `drawGame()` is called; this is responsible

QUICK TIP

This is the first of two halves of our final game. Everything works but doesn't look too polished quite yet, and the game mechanics are simple, but remember: we're setting the stage for next time.



Right The two ships for our game: our ship (left) and an alien ship (right)

only for drawing the elements of the game that may or may not have been changed. **`drawGame()`** could update different properties if we wanted it to, but that leads to messy code with multiple places where things can go wrong. At least this way, if something breaks in our game logic, we'll know that it most probably broke in **`updateGame()`** and not somewhere else. You'll notice that **`drawGame()`** is a good deal smaller than **`updateGame()`**; this is not necessarily because its job is simpler, but because all of the drawing onto our surface has been abstracted away to individual classes. Each object is responsible for drawing itself, but **`drawGame()`** has to tell them to do it.

Ships.py

On line 32 of **aliens.py**, we have the variable **`ship`**. This is where we create our player's spaceship. With it, we shall defend the Earth, our solar system and, yes, even the galaxy from the tyranny of evil alien kinds, because we are the Guardians of the… never mind, got a little carried away there. But what does this little variable do? Well, it instantiates our **`Player`** ship class that we imported from our **ship.py** file. If you take a look at **ship.py**, you'll see it's almost as big as **aliens.py**. That makes sense: after all, spaceships are complicated things. In our **ships.py** file, we have two different classes: our **`Player`** class (remember, class names usually start with a capital letter) and our **`Enemy`** class. The **`Player`** class is where all of the logic for moving, firing, drawing and damaging our own spaceship happens. We also keep track of the sound effects and images used by our spaceship as we play our game. Almost all of these functions will be called by the code in our **aliens.py** file, but our ship can call functions on itself too. For example, when we initialise our **`Player`** class, we call the **`loadImages()`** function inside of our **`__init__`** function with **`self.loadImages()`** so that we load our ship images straight away, rather than cause a delay when we need to draw our ship for the first time.

> "This is where we create our player's spaceship. With it, we shall defend the Earth, our solar system and, yes, even the galaxy from the tyranny of evil alien kinds"

Last, but certainly not least, we have our 'main loop' on lines 91–135. Now that our main loop is no longer responsible for updating or drawing our game events, it's mainly responsible for detecting the state of our mouse and timing events, like creating a new enemy spaceship after a certain amount of time. In a way, our main loop is still responsible for updating and drawing our game, in that it calls **`updateGame()`** and **`drawGame()`** at the correct time based on our game variables; the responsibilities have just been abstracted so our code is a little more decipherable to people who didn't write the code or read this tutorial.

Our **Enemy** class is much, much smaller than our **Player** class. Is this because it's less complicated? Nope. If you look at line 73 of **ships.py**, you'll see that when we call the **Enemy** class, we pass through the **Player** class as an argument. But why? What does this do? It's a really neat thing that classes do that we didn't get time to look over last time. When we instantiate a class, if we include the name of another class in its declaration when we actually write the code, it will get all of the properties and classes of the class that has been passed through. We do this because, despite being on opposite sides of our epic cosmic war of wits and lasers, at their core, a spaceship is a spaceship like any other but with a few tweaks here and there.

So, even though our **Enemy** class doesn't have **checkForHit**, **registerHit** and **checkForBullets** methods typed out, it still has those methods – it just gets them from **Player**. This lets us use code in different ways across multiple objects, but we can also overwrite some of those methods and add new ones as they're needed for our **Enemy** class. For example, our **Enemy** class has a **tryToFire()** function. Our **Player** class doesn't have this; only our **Enemy** class does. We can also set different values for the same variables in our **Enemy** class: our **bulletSpeed** value in **Enemy** is 10, whereas it's -10 in our **Player** class. And, of course, the image we're using for each type of ship is different.

Projectiles.py

Continuing our use of classes, we have our projectiles **Bullet** class in our **projectiles.py** file. Note that the latter isn't imported into our game in **aliens.py** but in **ships.py**, because our game doesn't need bullets – our ships do. Our **Bullet** class is far simpler than our two ship classes: we have only three methods and a variety of variables to affect and track each bullet – **move**, **draw**, and **loadImages**. How does each bullet know when it's hitting something? Because each bullet created in our game is stored in the **bullets** list in each of our ships, we use the **ships** class method **checkForHit** to see whether or not any of the bullets hit anything. There's no real reason for doing it this way; we could have each bullet be responsible for checking if it hit something, but it makes sense to have each ship keep an eye on whether the bullets it fired hit something.

Game events

Now we know what each file is responsible for in our latest game, we can start walking through how they all come together. As soon as our game is run, the global variables are set and all of the imports between lines 1 and 39 are applied. Let's take a look at our main loop starting on line 90 in **aliens.py**. The first thing we do on each loop of our game is set three variables: **timeTicks**, **mousePosition**, and **mouseStates**. The **timeTicks** variable keeps track of how many milliseconds have passed since we started the game. We can use this to set variables or create new objects after a set amount of time, as we do on line 128, where we create a new enemy ship after every 1.5 seconds of the game playing. **mousePosition** is where we store the position of the mouse in our game window. Remember, our mouse

> The first thing we do on each loop of our game is set three variables

Left The two different projectiles for our ship types: our projectile (top) and the alien projectile (bottom)

QUICK TIP

The sounds for this tutorial were created using BFXR (**bfxr.net**), a nifty little tool designed for creating sound effects that resemble those from games of times long past. Go and have a play with it!

Above The white boxes around our spaceships are a visual representation of the 'hit test' that we're running to check if a projectile has hit a ship

position is relative to the top-left of our Pygame window. We use the mouse to move our ship around in our game, so we need to keep a constant track of where our mouse is; storing our **`mousePosition`** at the start of each loop saves us a little time and typing if we need to check the position more than once every loop. **`mouseStates`** is where we save the 'state' of our mouse buttons – that is, which buttons are being pressed down and which ones aren't. We use the left mouse button to fire our weapons and start our game so again, having that information stored globally means we can check against one variable rather than calling **`pygame.mouse.get_pressed()`** multiple times.

Starting our game

Right after the first three variables, we come across an if-else statement. This will check whether or not our game has started; after all, we want to have a title and game-over screen, so we need to know when to show them. The first check on line 97 will see if we are running a game already. If so, it'll update and then draw the game (we'll go through those shortly); otherwise, we go to the next check in our if-else statement. On line 102, if our game hasn't started yet and it hasn't finished either, then we must have just started the program, so let's draw the start screen. On line 103, we blit our start screen image onto our game surface. This image has a Start button drawn on it, so next, on line 105, we check whether or not the left mouse button has been clicked and if it has, we check to see if the click occurred inside the button on line 107. If both of these conditions are met, our **`gameStarted`** variable is set to **`True`** and on the next loop, our game will start – time to kill the alien scourge!

Updating the game state

The game has started, the onslaught begins, but how do we go about saving our kind? Line by line, of course! Lines 41–75 of **aliens.py** contain our **`updateGame()`** method. Here, we update the position of our ships, the enemy ships, and fire our weapons if we click a mouse. On lines 45–49, we check whether or not our mouse is being clicked. If it is, we fire our weapon, but we don't want our guns to keep firing for as long as we hold down the button; we want to fire on each click, so we set the **`mouseDown`** variable to **`True`**. This way, we can be certain that we only fire once per click, not willy-nilly. When we fire our weapon, we play a laser sound effect. True, in space no-one can hear you scream, but in *Star Wars* it's infinitely cooler to have blaster sounds going off all around you. Much like when we add an image to our surface to see it drawn on our screen, we add our

sound to our mixer to have it played out through our speakers (on lines 31–33 of **ships.py**).

Next, on line 51, we set the position of the ship to match where our mouse is. We subtract half of the width of our ship from our mouse X coordinate, so the ship's centre aligns with our mouse pointer.

That's everything to do with our ship updated. Simple, eh? On to the enemy ships now. Unlike our valiant spaceship there are many, many enemy spaceships. We need to update the position and state of each one of them, so we create a loop on lines 57–72 to handle this. First, we move them. Our enemy spaceships aren't that sophisticated when they move; they're hell-bent on our destruction, so they fly straight at us in order to take a potshot. Next, the enemy spaceships will try to take a shot at us. Why 'try'? Well, our enemies are being controlled by Python, which can fire a darned sight quicker than you can. **`tryToFire()`** is called once per ship in every loop and gives our enemy a 1/100 chance of getting off a shot. That might sound like pretty slim odds for firing at all! But remember, our loop is run 60 times a second, which means that there's a roughly 50–50 chance each enemy will fire a shot every two seconds, so it's enough to keep our wits about us.

Lines 60–61 are where we check whether any shots our enemies have fired have hit us, and whether any shots we've fired have hit our enemies. The **`checkForHit`** function call does a couple of things. We pass through the thing we want to check that we hit; if we did, our code in **ships.py** on lines 48–49 will figure that out. If we did hit the enemy, or vice versa, our **`checkForHit`** function will call the **`registerHit()`** function on the object that we hit, which decreases the health value of our target. Our enemies have one life, whereas we have five. At the end of our **`checkForHit()`** function, we return **`True`** if the health of the ship is 0; that way, if our **`shipIsDestroyed == True`** or our **`enemyIsDestroyed == True`**, we can end the game or remove the enemy from our game.

Rather than removing our enemy straight away, we add its index in the **`enemyShips`** list to the **`enemiesToRemove`** list. Once we've worked out all of the enemies we need to delete, we iterate through the indexes in the **`enemiesToRemove`** list and delete them one at a time on lines 74–75.

Drawing our game

As noted previously, the **`drawGame()`** function is much smaller than the **`updateGame()`** one. This is because **`updateGame()`** has done all of the hard work for us. We don't need to worry about moving anything here – everything that needs to be updated, like positions and health, has already been taken care of before we get to **`drawGame()`**.

The first thing we draw with the latter is the background, because we want everything else to be drawn on top of it. We only have a single player spaceship, so we'll draw that first and then we'll draw all of the bullets that we've fired so far on lines 79–80. Next, we'll draw our enemies. Again, we'll do this in a loop because there's the possibility of there being more than one enemy, and we'll draw the bullets that each ship fires too. It wouldn't be much of an armada if there was only one enemy ship.

> We set the position of the ship to match where our mouse is. We subtract half of the width of our ship from our mouse X coordinate so the ship's centre aligns

Next time...

That's the first half of our game. We don't have much in the way of a game-over screen, so we'll make one of those. We'll program a UI for our health, add shields to our ships, and include some explosion effects when either our own ship or an enemy ship is destroyed. We'll also write some code that will create levels (waves) that you can customise to make each game behave any way you like.

Below If we don't listen for the 'mouse_up' event, our ship will be able to fire as quickly as Pygame can handle it, which doesn't make for a very interesting game, even if it is very pretty..

Aliens.py

```
import pygame, sys, random, math
import pygame.locals as GAME_GLOBALS
import pygame.event as GAME_EVENTS
import pygame.time as GAME_TIME
import ships

windowWidth = 1024
windowHeight = 614

pygame.init()
pygame.font.init()
surface = pygame.display.set_mode((windowWidth, windowHeight))

pygame.display.set_caption('Alien\'s Are Gonna Kill Me!')
textFont = pygame.font.SysFont("monospace", 50)

gameStarted = False
gameStartedTime = 0
gameFinishedTime = 0
gameOver = False

# Mouse variables
mousePosition = (0,0)
mouseStates = None
mouseDown = False

# Image variables
startScreen = pygame.image.load("assets/start_screen.png")
background = pygame.image.load("assets/background.png")

# Ships
ship = ships.Player(windowWidth / 2, windowHeight, pygame, surface)
enemyShips = []

lastEnemyCreated = 0
enemyInterval = random.randint(1000, 2500)

# Sound setup
pygame.mixer.init()

def updateGame():

  global mouseDown, gameOver

  if mouseStates[0] is 1 and mouseDown is False:
    ship.fire()
    mouseDown = True
  elif mouseStates[0] is 0 and mouseDown is True:
    mouseDown = False

  ship.setPosition(mousePosition)

  enemiesToRemove = []

  for idx, enemy in enumerate(enemyShips):

    if enemy.y < windowHeight:
      enemy.move()
      enemy.tryToFire()
      shipIsDestroyed = enemy.checkForHit(ship)
      enemyIsDestroyed = ship.checkForHit(enemy)

      if enemyIsDestroyed is True:
        enemiesToRemove.append(idx)

      if shipIsDestroyed is True:
        gameOver = True
        print "\n\n\nYou Died\n\n\n"
        quitGame()

    else:
      enemiesToRemove.append(idx)

  for idx in enemiesToRemove:
    del enemyShips[idx]

def drawGame():
    surface.blit(background, (0, 0))
    ship.draw()
    ship.drawBullets()

    for enemy in enemyShips:
      enemy.draw()
      enemy.drawBullets()

def quitGame():
  pygame.quit()
  sys.exit()

# 'main' loop
while True:

  timeTick = GAME_TIME.get_ticks()
  mousePosition = pygame.mouse.get_pos()
  mouseStates = pygame.mouse.get_pressed()

  if gameStarted is True and gameOver is False:

    updateGame()
    drawGame()

  elif gameStarted is False and gameOver is False:
    surface.blit(startScreen, (0, 0))

    if mouseStates[0] is 1:

      if mousePosition[0] > 445 and mousePosition[0] < 580 and mousePosition[1] > 450 and mousePosition[1] < 510:

        gameStarted = True

    elif mouseStates[0] is 0 and mouseDown is True:
      mouseDown = False

  elif gameStarted is True and gameOver is True:
          surface.blit(startScreen, (0, 0))
          timeLasted = (gameFinishedTime - gameStartedTime) / 1000

  # Handle user and system events
  for event in GAME_EVENTS.get():

    if event.type == pygame.KEYDOWN:

      if event.key == pygame.K_ESCAPE:
        quitGame()

  if GAME_TIME.get_ticks() - lastEnemyCreated > enemyInterval and gameStarted is True:
    enemyShips.append(ships.Enemy(random.randint(0, windowWidth), -60, pygame, surface, 1))
    lastEnemyCreated = GAME_TIME.get_ticks()

  if event.type == GAME_GLOBALS.QUIT:
    quitGame()

  pygame.display.update()

```

Language
>PYTHON

DOWNLOAD:
github.com/seanmtracey/Games-with-Pygame/tree/master/Part%209

Ships.py

```
01. import projectiles, random
02.
03. class Player():
04.
05.   x = 0
06.   y = 0
07.   firing = False
08.   image = None
09.   soundEffect = 'sounds/player_laser.wav'
10.   pygame = None
11.   surface = None
12.   width = 0
13.   height = 0
14.   bullets = []
15.   bulletImage = "assets/you_pellet.png"
16.   bulletSpeed = -10
17.   health = 5
18.
19.   def loadImages(self):
20.     self.image = self.pygame.image.load("assets/you_ship.png")
21.
22.   def draw(self):
23.     self.surface.blit(self.image, (self.x, self.y))
24.
25.   def setPosition(self, pos):
26.     self.x = pos[0] - self.width / 2
27.     # self.y = pos[1]
28.
29.   def fire(self):
30.     self.bullets.append(projectiles.Bullet(
    self.x + self.width / 2, self.y, self.pygame, self.surface,
    self.bulletSpeed, self.bulletImage))
31.     a = self.pygame.mixer.Sound(self.soundEffect)
32.     a.set_volume(0.2)
33.     a.play()
34.
35.
36.   def drawBullets(self):
37.     for b in self.bullets:
38.       b.move()
39.       b.draw()
40.
41.   def registerHit(self):
42.     self.health -= 1
43.
44.   def checkForHit(self, thingToCheckAgainst):
45.     bulletsToRemove = []
46.
47.     for idx, b in enumerate(self.bullets):
48.       if b.x > thingToCheckAgainst.x and b.x <
    thingToCheckAgainst.x + thingToCheckAgainst.width:
49.         if b.y > thingToCheckAgainst.y and b.y <
    thingToCheckAgainst.y + thingToCheckAgainst.height:
50.           thingToCheckAgainst.registerHit()
51.           bulletsToRemove.append(idx)
52.
53.     for usedBullet in bulletsToRemove:
54.       del self.bullets[usedBullet]
55.
56.     if thingToCheckAgainst.health <= 0:
57.       return True
58.
59.   def __init__(self, x, y, pygame, surface):
60.     self.x = x
61.     self.y = y
62.     self.pygame = pygame
63.     self.surface = surface
64.     self.loadImages()
65.
66.     dimensions = self.image.get_rect().size
67.     self.width = dimensions[0]
68.     self.height = dimensions[1]
69.
70.     self.x -= self.width / 2
71.     self.y -= self.height + 10
72.
73. class Enemy(Player):
74.
75.   x = 0
76.   y = 0
77.   firing = False
78.   image = None
79.   soundEffect = 'sounds/enemy_laser.wav'
80.   bulletImage = "assets/them_pellet.png"
81.   bulletSpeed = 10
82.   speed = 2
83.
84.   def move(self):
85.     self.y += self.speed
86.
87.   def tryToFire(self):
88.     shouldFire = random.random()
89.
90.     if shouldFire <= 0.01:
91.       self.fire()
92.
93.   def loadImages(self):
94.     self.image = self.pygame.image.load("assets/them_ship.png")
95.
96.   def __init__(self, x, y, pygame, surface, health):
97.     self.x = x
98.     self.y = y
99.     self.pygame = pygame
100.    self.surface = surface
101.    self.loadImages()
102.    self.bullets = []
103.    self.health = health
104.
105.    dimensions = self.image.get_rect().size
106.    self.width = dimensions[0]
107.    self.height = dimensions[1]
108.
109.    self.x -= self.width / 2
```

Projectiles.py

```
01. class Bullet():
02.
03.   x = 0
04.   y = 0
05.   image = None
06.   pygame = None
07.   surface = None
08.   width = 0
09.   height = 0
10.   speed = 0.0
11.
12.   def loadImages(self):
13.     self.image = self.pygame.image.load(self.image)
14.
15.   def draw(self):
16.     self.surface.blit(self.image, (self.x, self.y))
17.
18.   def move(self):
19.       self.y += self.speed
20.
21.   def __init__(self, x, y, pygame, surface, speed, image):
22.     self.x = x
23.     self.y = y
24.     self.pygame = pygame
25.     self.surface = surface
26.     self.image = image
27.     self.loadImages()
28.     self.speed = speed
29.
30.     dimensions = self.image.get_rect().size
31.     self.width = dimensions[0]
32.     self.height = dimensions[1]
33.
34.     self.x -= self.width / 2
```

FREQUENTLY ASKED QUESTIONS

NEED A PROBLEM SOLVED?

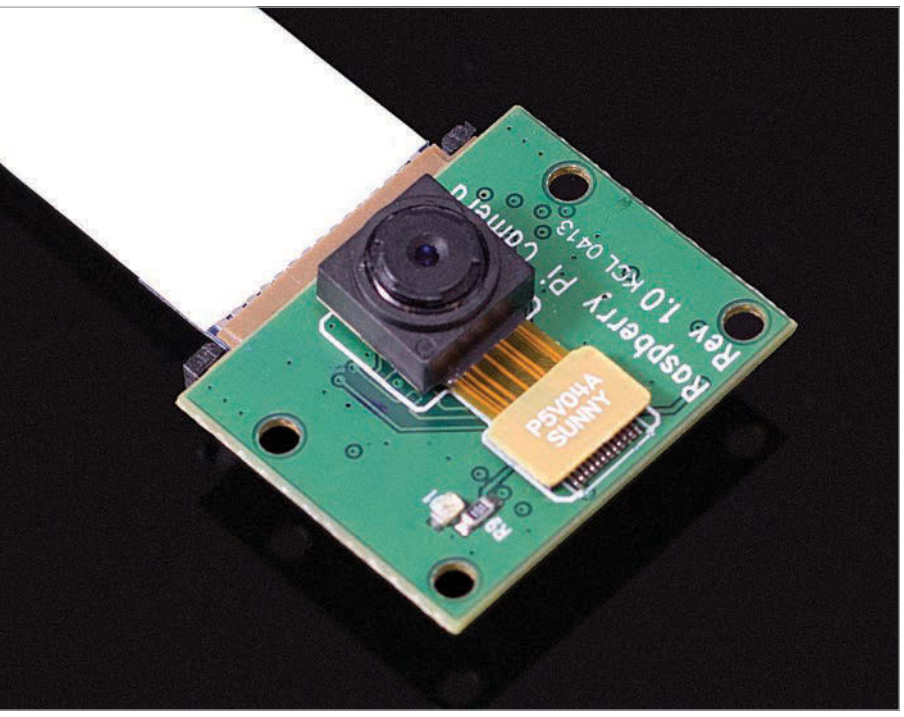
Email **magpi@raspberrypi.org** or find us on **raspberrypi.org/forums** to feature in a future issue.

Your technical hardware and software problems solved...

CAMERA

Can I only use the Pi Camera Module with the Raspberry Pi, or can I use other types of cameras?

The Camera Module is only one way to connect a camera to the Raspberry Pi; other methods will work as long as Raspbian supports the drivers for your camera, or you have some you can install. Most commonly, you'll find USB webcams hooked up to the Pi as they're readily available and fairly standard to use. One of the benefits of the Camera Module when it was released was how it freed up a USB port, especially when there were only two available on the Pi Model B. Aside from USB webcams, we've also seen people wire up DSLRs via a breadboard and the GPIO pins.



If I hook up a USB webcam, will I be able to use the same Camera Module Python code to control it?

No, unfortunately not. That code is very specifically designed to work with the Camera Module connected via the CSI port on top of the Raspberry Pi. However, you can still program it, although you may need to look up methods on how to control your specific webcam with Python or another programming language. In theory, you can do a similar range of functions as the Pi camera: take video, time-lapse photography, etc, with only the webcam's software as the limitation.

Is the other slot on the top of the Raspberry Pi for putting a second camera in?

No, that's a display port for adding a screen to the Raspberry Pi. You can only hook one Camera Module up to the Pi at a time, although that doesn't mean you can't add several more webcams via USB ports if you want to create some kind of surveillance system, 3D scanner, or DIY bullet-time effect creator.

Can I improve the resolution of the slow-motion video function?

Unfortunately not – video recording is limited compared to photo taking as it is; however, the 640×480 resolution of slow-motion video is a necessary limitation, as the camera and Pi have to work a lot harder to record the images and put them into a video file. There are ways around it, though; have a look in our 30 Minute Projects feature this issue to see how you can take a rapid burst of images and then combine them into a video for higher-resolution slow-motion video that can be filmed at an even higher frame rate than the standard function. It just takes a bit more manual processing from you.

FROM THE RASPBERRY PI FAQ
RASPBERRYPI.ORG/HELP

Does the Raspberry Pi have an official programming language?
The Raspberry Pi Foundation recommends Python as a language for learners. It also recommends Scratch for younger kids. Any language which will compile for ARMv6 (Pi 1) or ARMv7 (Pi 2) can be used with the Raspberry Pi, though, so you are not limited to using Python. C, C++, Java, Scratch, and Ruby all come installed by default on the Raspberry Pi.

Will the Raspberry Pi run Wine (or Windows, or other x86 software)?
In general, with most versions of the Raspberry Pi, this is not possible. Some people have put Windows 3.1 on the Raspberry Pi inside an x86 CPU emulator in order to use specific applications, but trying to use a version of Windows even as recent as Windows 98 can take hours to boot into, and may take several more hours to update your cursor every time you try to move it. We don't recommend it! As of summer 2015, a version of Windows 10 is available for use on the Raspberry Pi 2. This will be an entirely new version of the operating system, designed exclusively for embedded use, dubbed the Windows 10 Internet of Things (IoT) Core. It will not include the user interface ('shell') or the desktop operating system.

Will the Raspberry Pi run the Windows 8 ARM Edition?
No. Even if Microsoft decided to devote all its resources to getting Windows 8 on the Pi, it would not work. The Raspberry Pi lacks the minimum memory and CPU requirements, it runs on a version of the ARM processor that is not supported by Windows 8, it lacks the appropriate axis sensors... the list goes on and on. The Pi will not run Windows 8.

Will the Raspberry Pi run Android?
No. While a version of Android can be found in the forum, it is not stable enough for everyday use. There are no plans to continue working on it, as Android does not provide any enhancement to educational purposes that are not already fulfilled more readily with existing software – we see it as a platform for consumption, not creation.

THE MAGPI APP

Having trouble with *The MagPi* on the App Store or Google Play? Here are your most common questions answered:

How do I find *The MagPi* on Google Play or the App Store?
All you have to do is go to the search bar and type 'The MagPi' or 'Raspberry Pi' to find us.

I've subscribed to the digital edition and I can't sign in to restore my purchases. Please help!
Since your *The MagPi* purchases are linked to your Google or Apple accounts, there's no need to sign in at all. If you'd like to re-download your purchases on your current device, or make your purchases available on other devices, all you need to do is hit 'Subscribe' on the home screen, then 'Restore Purchases' on the next screen.

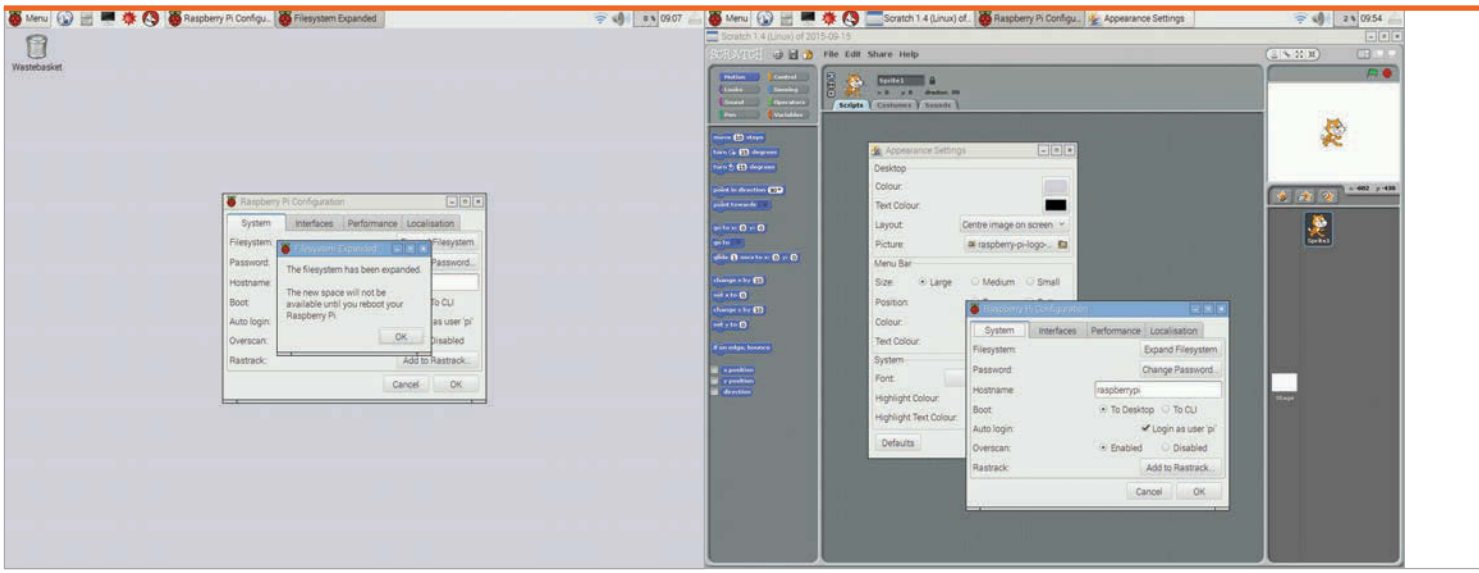
How can I search the digital magazine for keywords?
Finding direct references is really easy with *The MagPi* app – all you have to do is tap the screen to get the app's GUI to show, and then press the small magnifying glass icon in the top-right corner of the screen. Now, just type in your search term to find the relevant results.

Get it on Google play

Available on the App Store

raspberrypi.org/downloads

FREE (As in beer & speech)



RASPBIAN 8.0 'JESSIE'

A long time coming, is Raspbian Jessie the must-have update for the Raspberry Pi or can you stick with Wheezy?

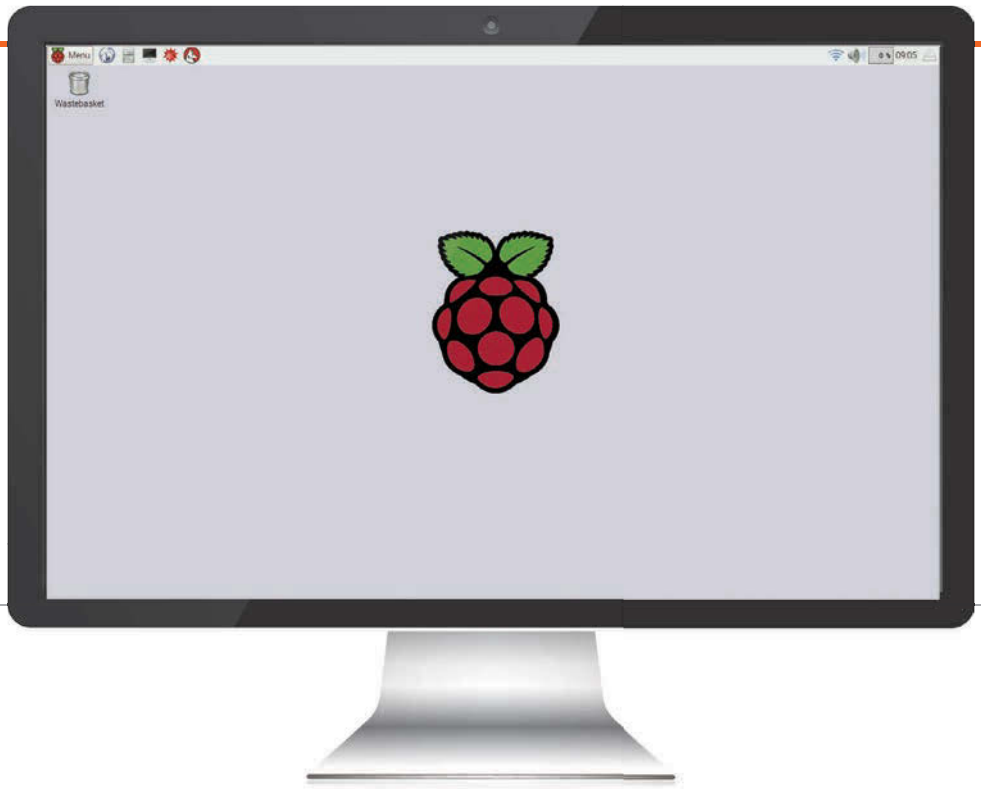
After a couple of years of Raspbian staying largely the same, we've recently seen a big effort to actually update the look and feel of the standard Raspberry Pi operating system. Sure, new bits of software have been added to the distro images previously, like Wolfram and the old Pi Store, but it was mostly the same old Raspbian running the LXDE desktop interface.

Over the past year, however, we have seen some major improvements and additions to Raspbian, most notably a new browser based on GNOME's Epiphany and a custom desktop environment built on top of the LXDE interface that was already there; both of these gave Raspbian a nicer look and feel, along with much improved usability.

Recently, the latest big update for Raspbian came about, involving a move to a new version of the Debian base system that it uses. The newer Raspbian 'Jessie' has been a while coming, and you can read a full breakdown of what's new in this version and why in our interview with the principal engineer behind it, starting on page 6 of this issue.

How does it actually perform, then? Well, largely the same. The move to Jessie brings with it a whole host of updated and new packages that make the experience a bit more in line with the current state of the software that you can install to the Raspberry Pi. LibreOffice and Claws Mail now come pre-installed, too. While this is nice, again it all feels much the same. There are no real speed improvements or optimisations and, luckily, no issues with performance on earlier Raspberry Pi models, so the experience of using it is almost no different.

This is good in a way: the Raspberry Pi Foundation is keen for people using the Raspberry Pi to not have to keep learning new interfaces, and also for its existing guides and lessons to continue to work without needing new screenshots and rewrites. This is also the reason the system now boots to the desktop by default – we're of the opinion that this is a good change for the majority of Raspberry Pi users, with the ability



Maker Says

"Raspbian is the Foundation's official supported operating system

Raspberry Pi Foundation

to get to the command line still available for when people are ready to give it a shot.

As well as the minor-but-not-really-a-change switch from CLI to desktop, there are some bigger differences. The config screen you used to get at the start of each knowledgeable computer users and makers anyway, so throwing all these extra options up at users who wouldn't need or necessarily understand them didn't really accomplish much in the first place. Again, they can access it at their own pace in a slightly more friendly

> Additions and changes have been made with usability and education in mind

Raspbian boot now has its own dedicated graphical tool, which works a lot better than the old raspi-config ever did. Everything is tabbed and much better presented, and the overclock options now know which version of the Pi you're using so that it doesn't try any funky overvolting. Although the config menu doesn't pop up on first boot anymore, the only real change we'd ever find ourselves making is expanding the file system. The config screen was really for more interface, while people who know what they're doing can get it sorted once they're in the desktop.

We love that you don't have to use **sudo** any more for accessing the GPIO pins, and while the new GPIO Zero is not included in the current build of the Jessie, having both together really opens up the GPIO header to a lot more people. Scratch can also now access the GPIO pins, so the whole thing has been made a lot easier for everyone on every level; and of course, your old programs still work as you expect even if you do install GPIO Zero. It doesn't have every single function you'd want programmed in, after all.

The only real issue we have with Raspbian Jessie is the size: at over 4GB, you'll definitely need an 8GB microSD card to get it on the Pi, something not everyone will have. There have been some efforts by the community to slim it down, so keep an eye out for those if that tickles your interest.

Really, Raspbian Jessie is best described as being more Raspbian. Additions and changes have been made with usability and education in mind, and most of them, while fairly minor, will have a decent impact on these areas.

Last word

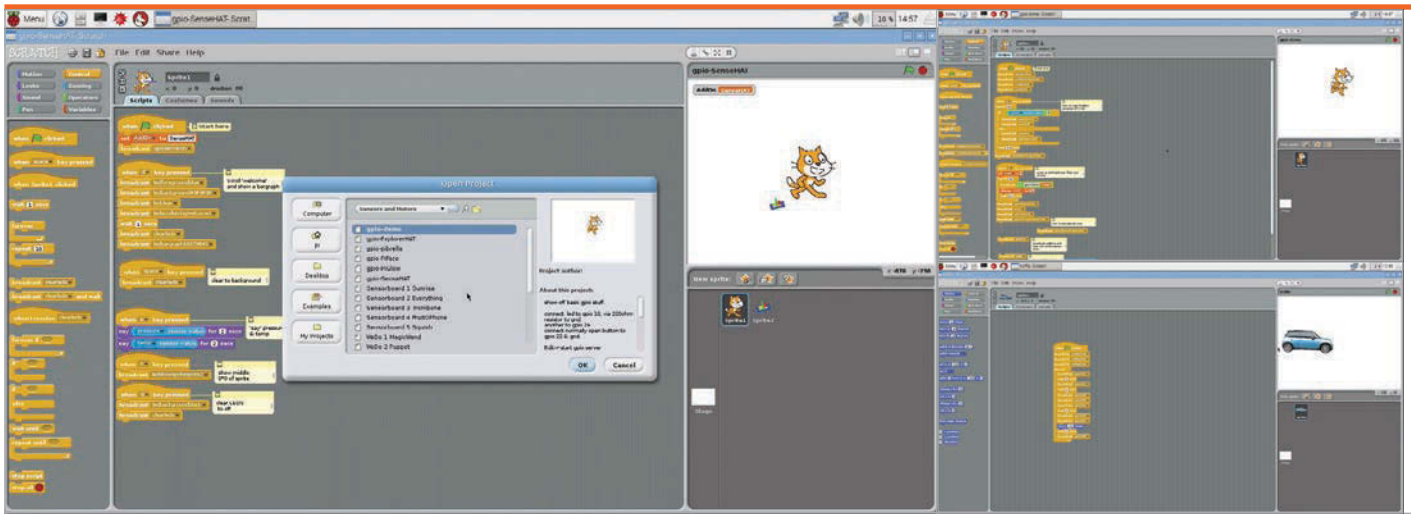
It's the same old Raspbian but slightly better here and there, albeit a little larger than it maybe should be. It reflects the mission of the Raspberry Pi Foundation, though, and is still the best Pi operating system.

★★★★☆

SCRATCH

scratch.mit.edu

FREE



SCRATCH

Tucked inside Raspbian Jessie is a revamped version of Scratch, full of new features. **Lucy Hattersley** takes it for a spin…

Related

FUZE BASIC 3.3

The FUZE Workstation is a great Pi-powered computer for schools and colleges, with a version of BASIC similar to that used by the BBC Microcomputer of yore. Most programmers started out with BASIC, and it remains a great way to learn text-based coding.

Free

www.fuze.co.uk

Scratch is one of the best ways to introduce young people to the joys of programming.

Developed by MIT specifically to help young people learn to code, Scratch is a visual editor with characters, known as sprites, and blocks that react to, and control, the sprites. The blocks click together in much the same way that code is written, and clicking the blocks together creates programs.

The Raspberry Pi Foundation's aim to build tiny and affordable computers for kids fits really well with Scratch's aim to help young people learn to code, so it's no surprise to see Scratch form part of the stock Raspbian operating system.

What you may not have noticed, however, is that the recent Raspbian Jessie update contains a revamped version of Scratch loaded with great new features.

The new version is based upon the same core edition of Scratch (version 1.4), but it now runs much faster – up to ten times as fast in some instances – and has native support for the GPIO pins. Let's handle the speed boost first.

The need for speed

While Scratch is developed by MIT, it's brought to the Raspberry Pi thanks to the hard work of a developer called Tim Rowledge (**rowledge.org**), and it's from his endeavours that we're getting Scratch's souped-up engine.

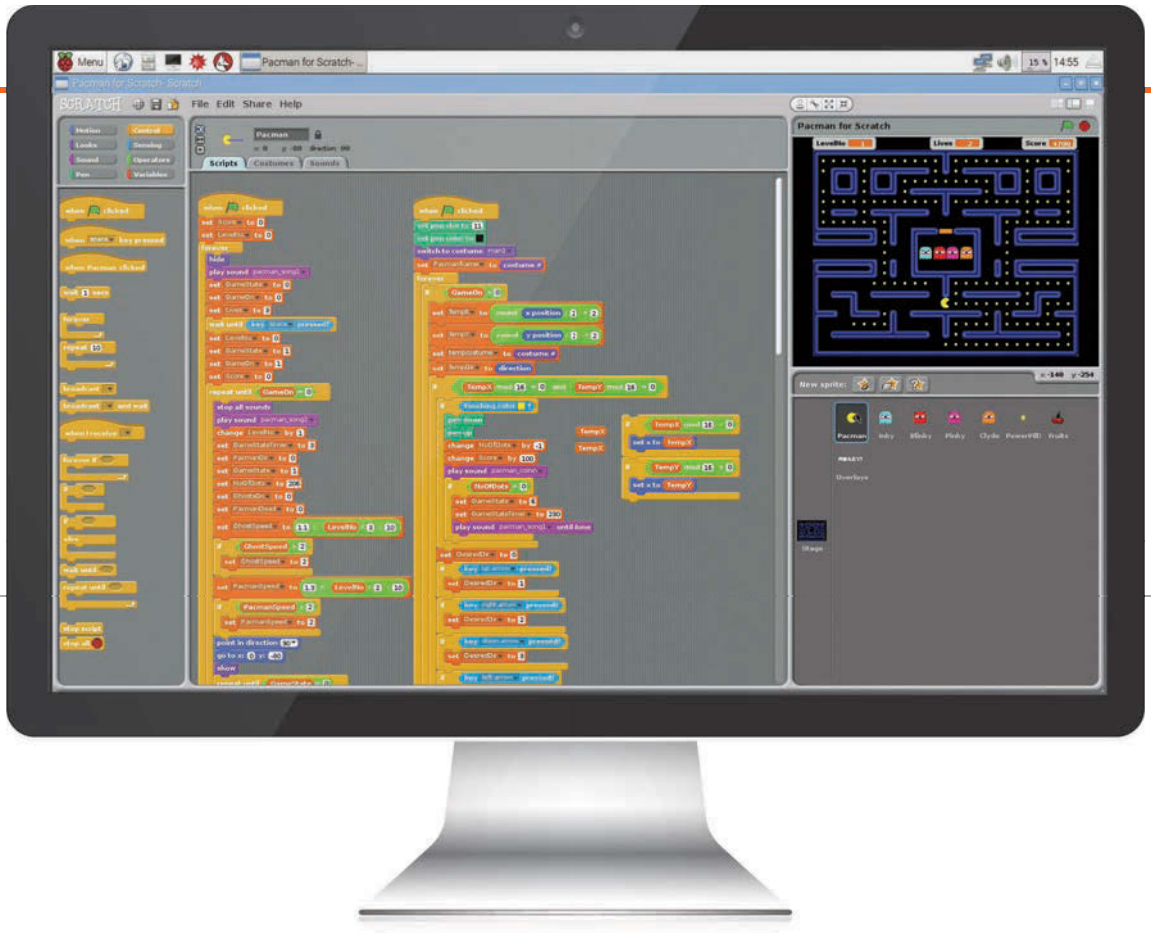
Scratch was originally developed in a language called Smalltalk, and it runs inside a Squeak virtual machine (VM). Tim has spent a lot of his time ensuring that Squeak runs on various ARM-based systems like the Raspberry Pi, mostly because he's a RISC OS fan. Tim says he spent time "rewriting some of the more egregiously ugly code, improving algorithms, tweaking VM configurations, and so on."

While direct speed comparisons depend on a lot of factors, a version of *Pac-Man* created in Scratch by Andrew Oliver is now running at a playable 30fps, up from 16fps on the Raspberry Pi model B and a mere 1fps on the original Pi.

This faster speed isn't just a nicety. The way Scratch gets kids coding is by recreating games and game-like environments. The improved Scratch performance prevents young people from hitting a performance limitation and switching off.

Using pins

If the speed boost wasn't enough, Scratch now incorporates GPIOServer so you can directly access the pins on your Raspberry Pi. It's been referred as a 'first pass' at GPIO implementation,



Maker Says

> Program your own interactive stories, games, and animations – and share your creations
>
> **Scratch**

and it's a bit clunky. Scratch has no GPIO blocks. Instead, you use the Broadcast block to set up pins, monitor input, and set them on or off. It follows the BCM standard, so you add a Broadcast block and enter **`config17out`** to set pin 17 to output, and then **`gpio17on`** and **`gpio17off`** to turn the pin on and off.

For input, you use **`config17in`** and then combine an If control block with an Operator and Sensing block to get a block that says: **`If GPIO 17 sensor value = 0/1`**. When you add GPIO pins, they appear in the sensor blocks.

We would really like an easier implementation for young coders to understand, but the good news is that there are lots of examples included with the installation, and tutorials are being developed or rewritten on the official Raspberry Pi website. So, you won't be short of assistance in getting to grips with it.

Teaching with GPIO

There is something truly joyous about combining Scratch with GPIO. It breaks the barrier between Scratch visual programming and physical hardware. You can create LED traffic lights that respond to sprites of cars, or build a joystick from switches to control on-screen characters. These are software and hardware projects that will inspire kids to get started.

As a bonus, many HATs are supported. We rather like the idea of hooking up Scratch characters to Pibrella, PiGlow, and Sense HAT devices.

Send in the clones

If there is one slight wobble, it's that this is still Scratch 1.4, rather than the newer Scratch 2.0. This is because MIT moved over to Adobe Flash for Scratch 2.0, and the offline app installs using Adobe Air. So we're stuck with the older version of Scratch until either Adobe starts to support ARM devices with Adobe Air, or MIT rewrites Scratch in a different language.

This isn't a big deal for the most part, but Scratch 2.0 does include a feature called cloning, via the 'Create Clone Of' block, which is very close to the object-orientated process of instantiation and inheritance. This is something we'd love to see in Scratch on the Raspberry Pi, but we'll take GPIO access over it any day.

Last word

The new version of Scratch breaks the barrier between hardware and software and creates an even more fun learning environment. And the speed boost makes more exciting programs playable. It's a shame that it's still based on version 1.4 rather than 2.0, but don't let that spoil things. This is a fantastic update.

sb-components.co.uk/products/mediapiplus

£39.99 / $60

Maker Says

"The new MEDIAPI+ augmented case for Raspberry Pi 2

SB-Components

MEDIA PI PLUS

A media centre case and remote for your Raspberry Pi: is it a necessity for a TV-connected Pi, though?

We all know at least one person who just has a Raspberry Pi for a media centre, running their TV on Kodi or even an ancient version of XBMC that does the job. One of the things that has been lacking from this equation for the longest time is a suitable case for a Pi hooked up to your television – a case that can slip in unnoticed under your TV, next to a Sky+ box and that Wii you haven't touched in years.

While the Media Pi Plus isn't the first one, or even the first Media Pi product, they're rare enough to highlight and discuss. In the Media Pi Plus's case, it's a re-release to fit the form factor of the Raspberry Pi B+ and the Pi 2. In aesthetic terms, it absolutely looks like something you'd put under your TV, specifically something like a Freeview box: it's unassuming, black, and with very little branding.

There is some construction required with the case, as a Raspberry Pi is not included. Popping the case open, you can see exactly how it works: it extends out and relocates a number of the Raspberry Pi's ports throughout a largely empty box. While this may seem slightly redundant, it does provide extra power to the USB ports, allowing for hubs to be connected. You can also connect an IR receiver directly to the GPIO ports, which works well with the media remote included, even if you have the ability to turn the Pi off but not on again with it.

It's really designed to be used with Kodi, which is why our version came with OpenELEC on a pre-formatted SD card, although other Kodi versions and offshoots (we're looking at you, OSMC) will work just fine. You may need to tweak some of the remote settings, but it will be fine nonetheless.

The best thing about the Media Pi Plus is that it's very cheap. Even if you factor in a Pi 2, it's just south of £70 for the whole thing, which is pretty great value for a media centre that will reliably serve you 1080p for a long time to come. Construction is a bit tricky and the case can feel a little flimsy, though if you aren't planning on flinging it around or otherwise treating it roughly, it's a perfectly serviceable way to make your Pi media centre just that bit easier to use.

Related

MODMYPI MODULAR CASE

The modular case can be modified to fit in better with your lounge, revealing only the ports you need to connect to a TV.

£5.99 / $9

bit.ly/1M4gQTi

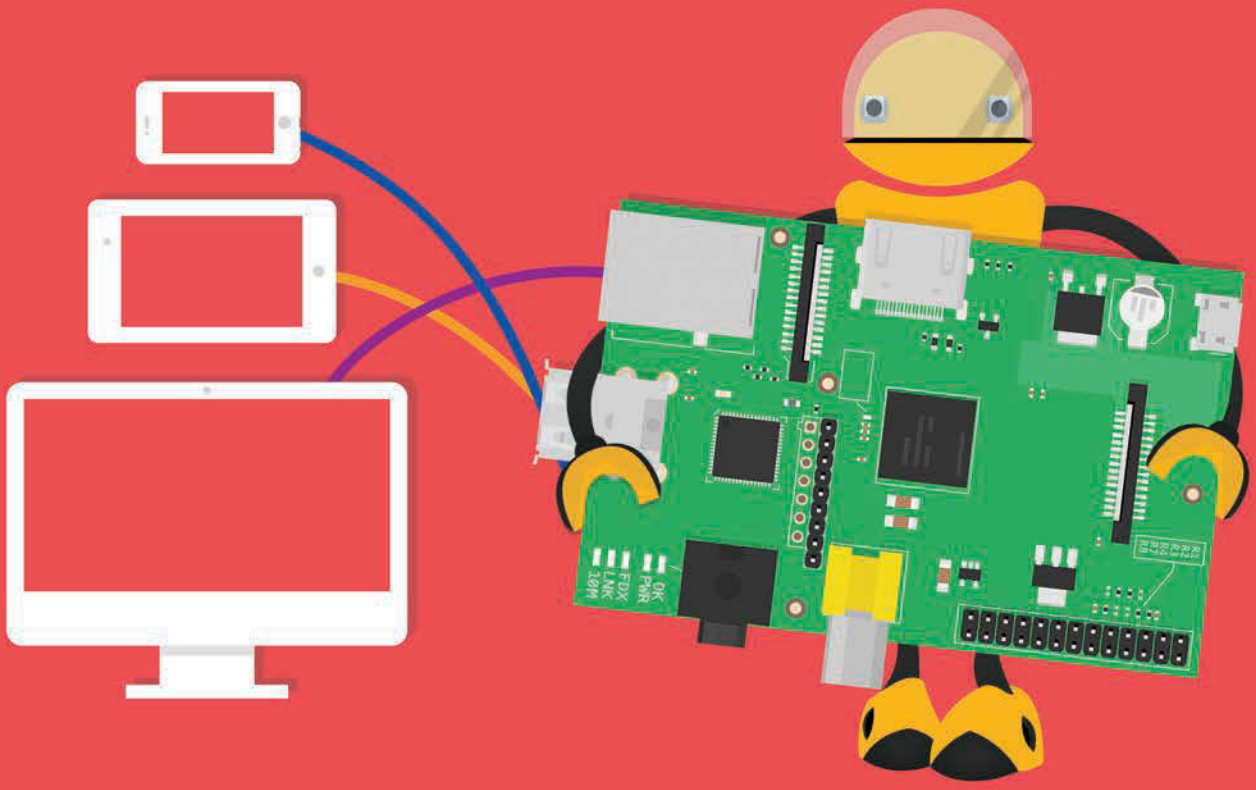
Last word

The box is a bit bulky in relation to the Pi's size, but it does a good job of making the small jump needed to get the Pi better suited for powering a TV.

★★★☆☆

VNC® now available for RASPBERRY PI!

RealVNC has released VNC for Raspberry Pi, which means you can now connect to your Pi from any Windows, Mac or Linux computer, or iPhone, iPad or Android device!



Check out our video tutorial to see how you can set up VNC Server on your Raspberry Pi.

Watch the video here: **https://www.youtube.com/user/RealVNCLtd**

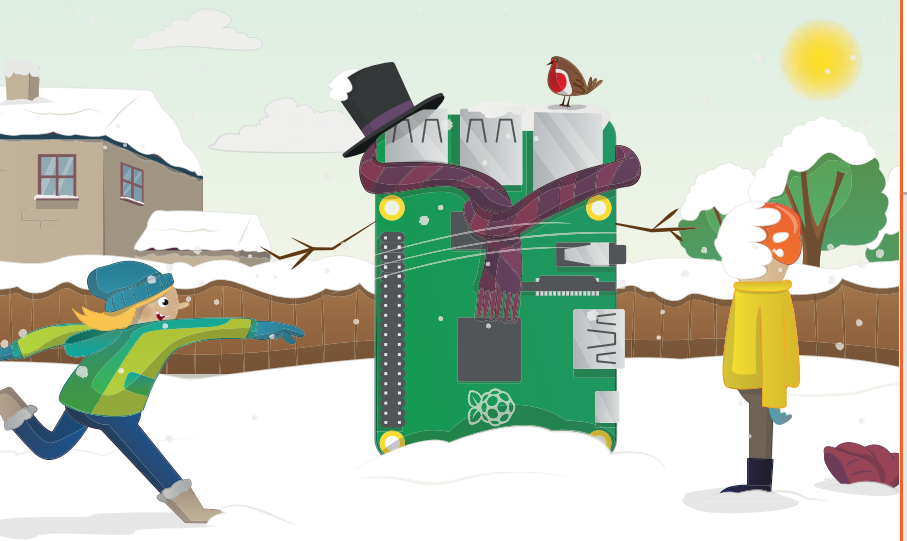
REAL VNC

Download VNC: www.realvnc.com/download/
Getting connected: www.realvnc.com/products/vnc/raspberrypi/
For more information contact sales@realvnc.com

STUFF YOUR STOCKINGS WITH RASPBERRY PI!

We've got you an early gift guide so you can start shopping for the techie in your life. Or just you.

It's already November. You've had to turn the heating on, you've gained an hour of sleep, and you're finally able to bring out the big winter coat. It's this time of year that we're constantly reminded that something big is coming in December: *Star Wars*. Also, Christmas. We like to get our shopping done early here, especially when the post starts getting a bit clogged up. So, straight from our Christmas list is the stuff that we think is the must-have Raspberry Pi gear to get and play around with on Christmas Day!



bit.ly/1hNMjko

£3

LEGO MICRO METAL GEARMOTOR ADAPTORS

There are a few ways to connect a Raspberry Pi up to Lego, Brick Pi being a good example off the top of our heads. That requires a bit of Lego Mindstorms, though, which is another layer on top of whatever code you'd want to do on the Pi. Great, but maybe not for everyone. This Lego axle adaptor works with tiny micro metal gearmotors, specifically the ones that Pimoroni sells at least, so you can build a Lego vehicle completely with Lego wheels that is powered solely by the Raspberry Pi's motors.

We love this. While technically you could always just use a Lego chassis with normal motors and wheels, this way you can use all of the new Lego kits you got at Christmas to create a supercharged monstrosity to chase down some mince pies.

bit.ly/1hNITxX

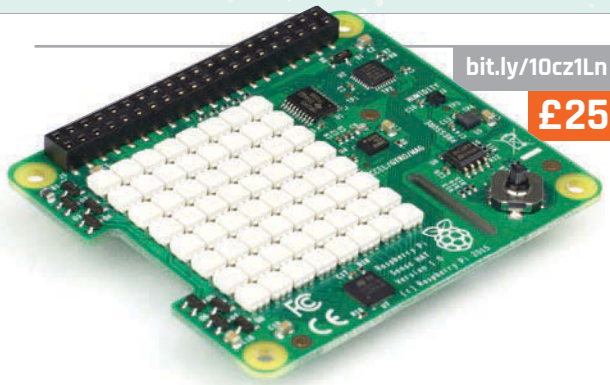
£17

CAMJAM EDUKIT #3

What exactly is in this new EduKit that can help you build a robot? Well, it has a couple of pre-wired motors that come with a motor control board, a battery holder, a line sensor, an ultrasonic sensor, a ball castor wheel, some custom red wheels for the motors, and other bits and bobs to get it all wired up.

Naturally, you will require a Raspberry Pi for it as well, although chances are if you're reading this magazine and are aware of these EduKits in the first place, that shouldn't be a problem.

The kits are out at the beginning of November, very probably by the time you read this article. We're not sure if the EduKit robots will be able to stand up to any of Pi Wars' house robots, but it will be very interesting to find out!



bit.ly/10cz1Ln

£25

SENSE HAT

We've talked about the Sense HAT a lot over the last couple of issues, but it's only because we've been having a lot of fun with it. As well as the ability to perform some fairly basic environmental measurements, such as temperature, humidity, and air pressure, it features a suite of motion sensors, a joystick, and a neat, fully programmable 8×8 LED display.

It's nice and cheap, easily installed onto the Raspberry Pi and from there you can program it with Python to make use of any of its inputs. And, of course, it's being used in space, which makes it extra cool. For starters, you can take a look at our pixel-art tutorial using Sense HAT later on in the mag, and check back in next issue for how to turn one into the best Christmas tree ornament you've ever seen.

bit.ly/1M2ZUCq

£2

RASPIO PORTSPLUS

This is a very simple little component: it slots over the 40-pin GPIO port of the Raspberry Pi A+, B+, and 2 and gives you a full rundown of what each pin can be used for, without getting in the way of any wires being plugged into the pins. It's very handy if you're not sure where the grounds are – and if you need to use the actual GPIO numbers instead of the pin numbers, you can flip it over to see what they should be.

This is especially good timing as a gift now that GPIO Zero is out, allowing you to know exactly where to plug components in and how to tell the code where they are without having to count each individual pin (or pair of pins) while squinting at a diagram on your phone.

bit.ly/1NRk6rq

£90

PICADE CONSOLE

We actually prefer this over the actual Picade that was Kickstarted all that time ago, mainly because we like the idea of a better version of those crap 17-in-1 arcade sticks you get in John Lewis or Next around Christmas time. Plug into your TV and play thousands of games on it using the genuine (pronounced 'gen-you-wine', with emphasis on each syllable) arcade machine parts to control them.

It's a bit pricey compared to the other items here, but it's a high-quality product that is an excellent way to get a MAME and retro console machine into your living room using Raspberry Pi. The arcade buttons are modifiable and upgradeable, you can plug in standard USB controllers, and the selection of games is frankly more than you'd ever be able to play through.

bit.ly/1LlloXU

£15

SIDEKICK: POPULAR COMPONENT KIT

Giving the gift of building and making with a Raspberry Pi at Christmas is great, but to do this you need a few very specific items to get someone started. Wires, breadboards, a few resistors, and a selection of components can go a long way, and while figuring out what to buy and where to buy them can be a bit tedious and confusing for someone starting out or trying to sort out a gift box of them, this Sidekick kit has you covered.

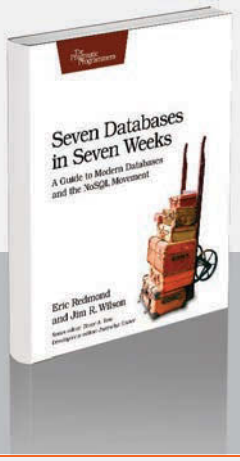
Presented in a little carry case so you can take it with you, the Sidekick comes with tons of little, basic electronic components, switches, motors, buzzers, and more to start you off on the path of the maker. Use it with tutorials in this mag, GPIO Zero, or start looking up stuff online; there's a fantastic selection out there to make the best use of this kit.

RASPBERRY PI BESTSELLERS

SEVEN WEEKS

Pragmatics' Seven Weeks series compares solutions in hot topics for programmers

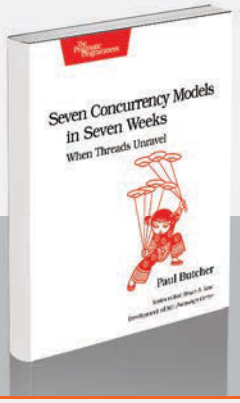
SEVEN DATABASES IN SEVEN WEEKS



Author: Eric Redmond & Jim R Wilson
Publisher: Pragmatic Bookshelf
Price: £23.50
ISBN: 978-1934356920
bit.ly/1eLiCzB

A practical dip into NoSQL databases: Redis, Neo4J, CouchDB, MongoDB, HBase, Riak (after PostgreSQL for comparison). The easiest way to quickly gain enough experience to make hands-on comparisons.

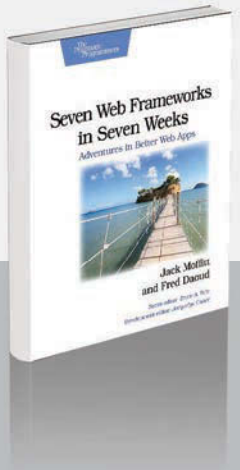
SEVEN CONCURRENCY MODELS IN SEVEN WEEKS



Authors: Paul Butcher
Publisher: Pragmatic Bookshelf
Price: £25.50
ISBN: 978-1937785659
bit.ly/1L5FItS

As multiple cores proliferate in the GPU as well as the CPU, Butcher tackles the competing techniques for parallelism and concurrency. Well written, and strongly grounded in Clojure.

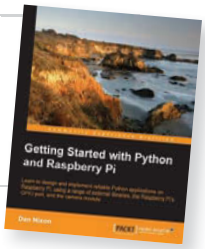
SEVEN WEB FRAMEWORKS IN SEVEN WEEKS



Authors: Jack Moffitt & Frederic Daoud
Publisher: Pragmatic Bookshelf
Price: £25.50
ISBN: 978-1937785635
bit.ly/1VEahMc

A fascinating and enjoyable journey: explore minimalism, composition, static typing, state machines, declarative syntax, and other approaches to web programming, using Sinatra, CanJS, AngularJS, Ring, Webmachine, Yesod, and Immutant.

GETTING STARTED WITH PYTHON & RASPBERRY PI



Author: Dan Nixon
Publisher: Packt
Price: £25.99
ISBN: 978-1783551590
bit.ly/1KZemHZ

We get to review many good and interesting Python tutorials, but a book placing Python learning within the context of programming the Raspberry Pi is always going to have an advantage: building quickly towards your own projects on the Pi is a great motivator. Nixon's book makes the link between basic Python learning and our favourite small board computer, even in the basics – such as treatment of numerical variables to best fit GPIO values.

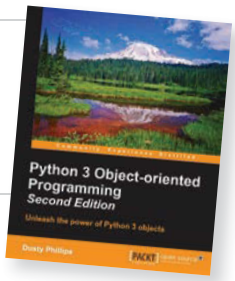
Nixon excuses his use of Python 2.7: "this has the widest library support and is still the default Python version on many operating systems." We suggest it's time to drop Python 2 for newcomers.

Nevertheless, the bulk of early chapters are a good Python introduction without distorting the lessons to be overly Pi-focused, giving a good grounding in control flow and data structures – procedural with a hint of functional flavour.

Object orientation is then introduced as a concept in a chapter also covering threads and locks. The peculiar areas of focus continue with a look at packaging in Python, but then the book turns to GPIO and the Camera Module. Data parsing and interfaces round out a useful introductory book, which will give confident learners enough immersion in Python to move onto their own projects.

Score ★★★★☆

PYTHON 3 OBJECT-ORIENTED PROGRAMMING



Author: Dusty Phillips
Publisher: Packt
Price: £31.99
ISBN: 978-1784398781
bit.ly/1LinBQP

A decade ago, Python was an overlooked scripting language and object-oriented programming (OOP) was introduced via Java (kitchen sink attached), Smalltalk (in academia at least – and subsequently powering Scratch 1.x), or even C++ (ouch). Luckily for learners of today, Python is widely accepted, and as a multi-paradigm language with strong OOP leanings it makes an ideal introduction to OOP. Phillips's book builds on basic Python knowledge you may have picked up with simple procedural scripts for the Pi, for example, and gives you the full object-oriented treatment. Starting, logically, with objects and classes, this leads onto inheritance; multiple inheritance is discussed (take that, Java) and then the first case study. Each chapter is rounded off with a useful case study and practically oriented exercises, often generating further thoughts on the topic to be carried to the reader's own work.

From attribute accessibility to design patterns, Phillips gives the essence of OOP in the context of Python's way of doing things – along the way stretching best practice a little, as some places aren't a comfortable fit for regular expressions – but he keeps pushing the reader to consider each bit of knowledge in the context of their own code, making for a very interactive book.

Score ★★★★☆

MAKE: THE BEST OF, VOLUME 2

Author: The Editors of Make
Publisher: Maker Media
Price: £19.99
ISBN: 978-1680450323
oreil.ly/1hpWhbr

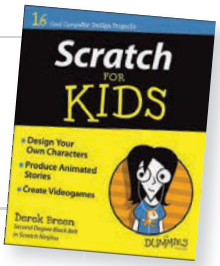
Want to be inspired? Pick a project from the contents page and start reading. We'd recommend something from Chapter 6: Music and Audio, which includes Cigar Box Guitars, a Laser Harp, and a Solar Xylophone. Having been inspired, you may find the slightly macho introductory section on building and equipping your workshop easier going, and be more forgiving of the jargon-laden text.

Inspiration can be found on almost every page, as this is a distillation of eight years of *Make* magazine since the first *Best Of* book was published. Projects are grouped by chapters such as Robots and Drones, Microcontrollers and Microcomputers, and Fun and Games. That middle one includes many interesting Arduino projects including sound synthesis, as well as turning a Pi into a Tor proxy for anonymous browsing, or an FM transmitter.

Projects range from the mundane but useful (audio amplifiers, lost screw finder), through difficult to classify (a Geiger counter), to pure fun (amazing light-up shoes). One of the most compelling, which few can read without feeling the urge to build, is The Most Useless Machine: flick the switch, and an arm reaches out from a door and turns the switch back off. That's all it does. Strangely compelling, eh?

Score ★★★★☆

SCRATCH FOR KIDS



Author: Derek Breen
Publisher: Wiley
Price: £21.99
ISBN: 978-1119014874
bit.ly/1FSU6Jm

Delete the cat. That's Breen's first instruction. Scratch is for everyone, not just younger children who like cute cats. But Breen has written a book that will get young and old quickly coding and learning – creating a *Flappy Bird* clone in the first chapter, such is the power of Scratch.

The introduction, very much in the spirit of Italo Calvino's *If On A Winter's Night A Traveller* – but with stronger overt humour – draws you straight in, and you'll only put the book down long enough to switch on your computer and open Scratch. The author's humour and personality bring the tutorials to life, but the useful level of detail ensures that the lessons stay with the reader much longer.

Split into three sections – Designer, Animator, Game Developer – *Scratch for Kids* is very strong on graphics, with the vector and collage chapters being a great end to the first section. The Animation section builds into a great way to teach visually oriented children how to code. Attention to detail, good graphics and great writing characterise a game section that takes four classic arcade games and once more sneaks in learning by osmosis. Unreservedly recommended for kids of all ages – including grown-up ones!

Score ★★★★★

ESSENTIAL READING: DATA

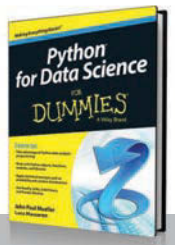
Data – big or otherwise – is a growing concern for coders in any language

Clean Data

Author: Megan Squire
Publisher: Packt
Price: £29.99
ISBN: 978-1785284014
bit.ly/1FWlLJh

Practical strategies to quickly and easily bridge the gap between the data we want and the data we have.

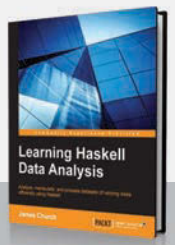
Python Data Science for Dummies



Author: John Paul Mueller & Luca Massaron
Publisher: Wiley
Price: £21.99
ISBN: 978-1118844182
bit.ly/1VEcYO1

A comprehensive introduction to practical data science using Python. Good for Python learners without being patronising.

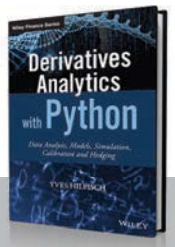
Learning Haskell Data Analysis



Author: James Church
Publisher: Packt
Price: £22.99
ISBN: 978-1784394707
bit.ly/1RtmVxc

Haskell-based intro to dealing with data that doesn't neglect its mathematical nature. Practical and educational.

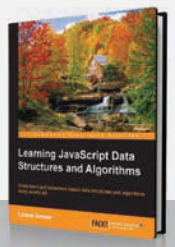
Derivatives Analytics with Python: Data Analysis, Models, Simulation, Calibration and Hedging



Author: Yves Hilpisch
Publisher: Wiley
Price: £60.00
ISBN: 978-1119037996
bit.ly/1HzDGSG

Growing use in the financial community has given us some useful tools for data analysis in Python.

Learning JavaScript Data Structures and Algorithms



Author: Loiane Groner
Publisher: Packt
Price: £27.99
ISBN: 978-1783554874
bit.ly/1MYzqzd

Beginner-friendly introduction to data structures and algorithms, using JavaScript. Comprehensive, approachable, excellent.

RASPBERRY JAM EVENT CALENDAR

Find out what community-organised, Raspberry Pi-themed events are happening near you...

PUT YOUR EVENT ON THE MAP

Want to add your get-together? List it here:
raspberrypi.org/jam/add

1 BIRMINGHAM RASPBERRY JAM

When: Saturday 31 October
Where: Library of Birmingham, Broad Street, B1 2ND
bit.ly/1jT8Opp
The Birmingham Picademy@Google team are hosting a Raspberry Jam at Google Digital Garage.

2 ROANOKE RASPBERRY JAM

When: Saturday 31 October
Where: 1327 Grandin Rd. SW, Roanoke, VA, USA
bit.ly/1PzW8Bd
There will be new projects and topics to discuss. Bring your favourite Raspberry Pi project for show-and-tell!

3 PRESTON RASPBERRY JAM

When: Monday 2 November
Where: Media Innovation Studio, Media Factory Building, Preston, UK
bit.ly/1W5Ao4c
A fun day out for all, with lightning talks, demos and hands-on time with the Raspberry Pi.

4 LEEDS RASPBERRY JAM

When: Wednesday 4 November
Where: Swallow Hill Community College, Leeds, UK
bit.ly/1kq0Wwc
Let's bring people together to discover the exciting potential of the Raspberry Pi.

5 NORTHERN ISLAND RASPBERRY JAM

When: Saturday 14 November
Where: Farset Labs, Belfast, UK
bit.ly/1K5ihnJ
The Northern Ireland Jam is aimed at complete beginners, with more complicated challenges for those with some previous Pi experience.

6 COTSWOLDS RASPBERRY JAM

When: Saturday 28 November
Where: University of Gloucestershire, Cheltenham, UK
cotswoldjam.org
The event will run between 1-4pm at the University of Gloucestershire Park Campus. Bring along your Pi.

7 CODERDOJO HAM MINI-JAM

When: Saturday 5 December
Where: Ham Close (off Ashburnham Road) Richmond, UK, TW10 7PL

coderdojoham.org

Each dojo has a variety of activities on offer including app creation, Pi projects, Python , HTML, and CSS.

8 PI WARS

When: Saturday 5 December
Where: Cambridge Computer Lab (William Gates Building), Cambridge, UK

piwars.org

The second annual roboteering event has arrived! Come along and join in the competition, or just watch the fun!

DON'T MISS: BARNES & NOBLE MINI MAKER FAIRE

When: Saturday 7 November **Where:** Various Barnes & Noble outlets

If you're a tech enthusiast, crafter, educator, tinkerer, hobbyist, engineer, science club member, author, artist, student, entrepreneur, or maker of any kind, you might want to head to your local Barnes & Noble. They're getting together in stores to learn from each other, hear from the experts, and work on projects. Visit **bit.ly/1NniPoR** to find a local Barnes & Noble that's taking part in this exciting event, or learn more at **bit.ly/1X6JYRE**.

TAKE US ANYWHERE

SAVE 45% with a Newsstand subscription (limited time offer)

FREE: ALL 30 ORIGINAL ISSUES NOW INCLUDED!

The MagPi Magazine

Available now

for smartphones & tablets

Available on the App Store

Get it on Google play

Subscribe from

£2.29 rolling subscription or **£19.99** full year subscription

Download it today - it's free!

- Get all 31 legacy issues free
- Instant downloads every month
- Fast rendering performance
- Live links & interactivity

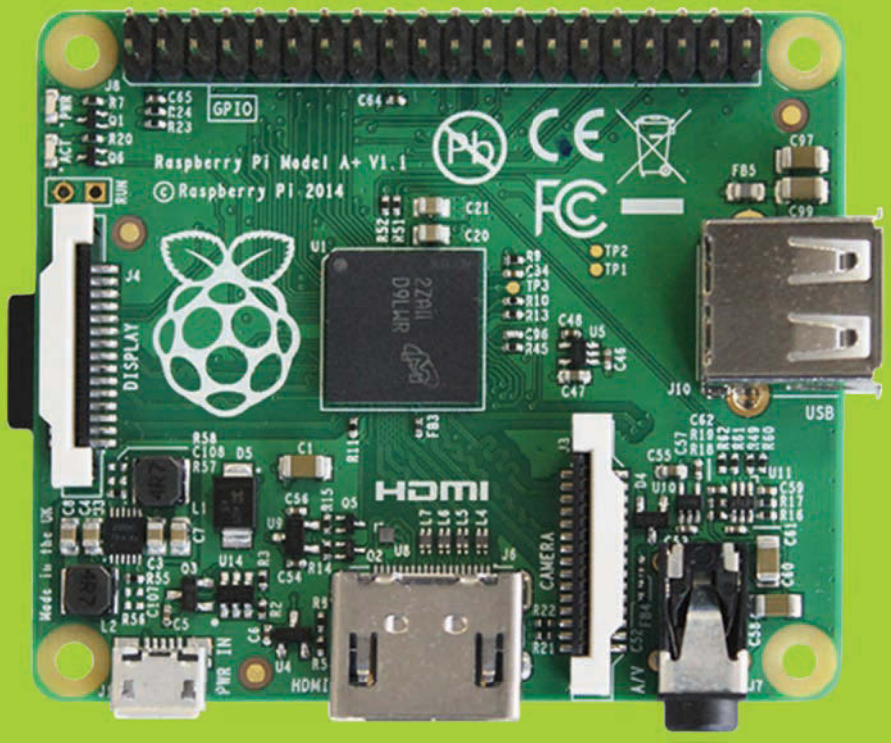
In association with:

MODMYPI

10 RASPBERRY PI Model A+s MUST BE WON!

WHAT RASPBERRY PI PLANS DO YOU HAVE FOR THE HOLIDAYS?

Tell us by 23 November for your chance to win!



How to enter:

Simply email **competition@raspberrypi.org** and let us know what Raspberry Pi-related projects or plans you have over the holidays!

Terms & Conditions

Competition closes 23 November 2015. Prize is offered worldwide to participants aged 18 or over, except employees of the Raspberry Pi Foundation, the prize supplier, their families or friends. Winners will be notified by email after the draw date. By entering the competition, the winner consents to any publicity generated from the competition in print and online. Participants agree to receive occasional newsletters from The MagPi magazine (unless otherwise stated upon entry). We don't like spam. Participants' details will remain strictly confidential and won't be shared with third parties. Prizes are non-negotiable and no cash alternative will be offered.

SKYCADEMY IN THE WILD

In August of this year, The Raspberry Pi Foundation provided a course for 24 educators in high-altitude ballooning. Here, one of them puts his training into action...


[01]


[02]

Stephen Brown is an assistant headteacher at Bourne Grammar School, and a passionate advocate of computer science and engineering in education. After coming to Cambridge in August to attend the Raspberry Pi Foundation's Skycademy course, he returned to Bourne eager to put a balloon into near space but committed to the idea that his students would plan the mission, build the payload, and launch the balloon independently.

He assembled four teams of students: the computer science team would set up and program the Pi's software, the engineers would put together the payload, the launch team would send the balloon to the stratosphere, and the chase team would track and retrieve the payload. He gave the students his documentation on how to run a successful launch, then let them get on with it.

"My favourite part was working on this project as a team with other students. It was good fun solving the problems and challenges we faced whilst working on it," says Andrew, who led the computer science team.

The students had two weeks to organise themselves, with a three-day launch window provided by the Civil Aviation Authority as a hard deadline that they had to meet.

The weather on launch day couldn't have been better, as the entirety of the school's Year 8 students assembled on the field, ready to watch the launch.

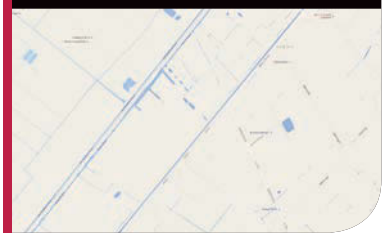
"Two weeks of lunchtime programming had gone into this and everyone was really nervous, as multiple things could have gone wrong," says Fabio, who watched his carefully assembled payload being tied to the helium balloon.

Fellow pupil Sohail was in charge of the launch preparation itself. He ensured that the flow of helium into the balloon remained steady, as he nervously tried to keep the fragile latex from coming too close to anything sharp, and carefully gauged when the required lift volume had been achieved.

With the balloon inflated, Patrick took control, ensuring the payload was securely attached, and carefully cable-tied the neck of the balloon to ensure it remained inflated.

BOURNE GRAMMAR SCHOOL

TOWN:
Bourne, Lincolnshire
DATE OF LAUNCH:
9 October 2015
RECOVERY AREA:
Near Welney, Norfolk
bit.ly/1LaBqnh




[03]

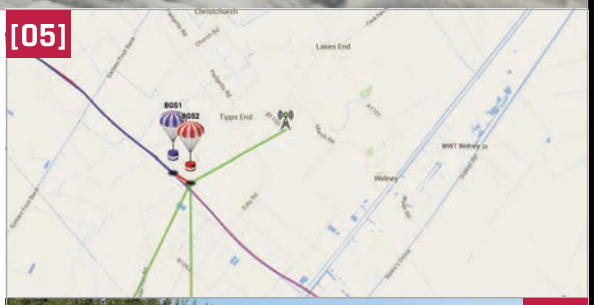

[04]

Lastly, it was up to Alex to launch the balloon. He stood before the expectant audience, letting the string feed through his fingers as the balloon ascended, and then finally releasing the payload and watching as all their hard work rapidly drifted upwards, now at the mercy of the atmosphere.

The chase team quickly jumped into the back of the school minibus and, with satnavs and radios blaring, they barked instructions to their driver, Systems & Control teacher Kevan Wackett. The original projections had the balloon scheduled to return to Earth somewhere near RAF Lakenheath, and there were visions of RAF intercept jets being launched to bring down the UFO. Fortunately, the course updated and the chase team headed for Welney, with the assistance of some expertly executed U-turns from Mr Wackett.

The payload came down in a field between two rivers, which required a two-mile cross-country hike to retrieve it. Half the students on the chase team had come prepared for a muddy trek, although two of the sixth-formers had missed the previous day's lunchtime meeting and looked more suited to a business interview than a brisk walk in the country.

The payload was retrieved without a hitch, safe and sound in its polystyrene box, having endured a 26km ride into the upper atmosphere, and being watched over by a herd of curious cows.


[05]


[06]


[07]


[08]

[01] THE RELEASE
At nine o'clock the balloon is launched, while the four teams of students and the whole of Year 8 watch their handiwork begin its incredible journey into the upper atmosphere.

[02] FIRST AERIAL PHOTOGRAPHS
A fantastic view of Bourne and the surrounding Lincolnshire countryside is captured as the balloon begins its ascent towards an overcast sky.

[03] ABOVE THE CLOUDS
The balloon breaks free of the clouds by 9:30.

[04] THE EDGE OF SPACE
The balloon has made it to the edge of space.

[05] THE DESCENT
The tracker has the payload predicted to land a little too close to a river. The tracking team back at school keep their fingers crossed that all will be well.

[06] THE RETRIEVAL
The chase team assemble, ready to retrieve the payload. Two of the students have hacked together wellington boots from bin liners, as they prepare to cross the muddy fields.

[07] FOUND IT
A relieved Alex finds the payload, intact and still transmitting telemetry and images.

[08] THE JOURNEY HOME
With the payload sitting on the minibus dashboard, the team make their way back to school.

YOUR LETTERS

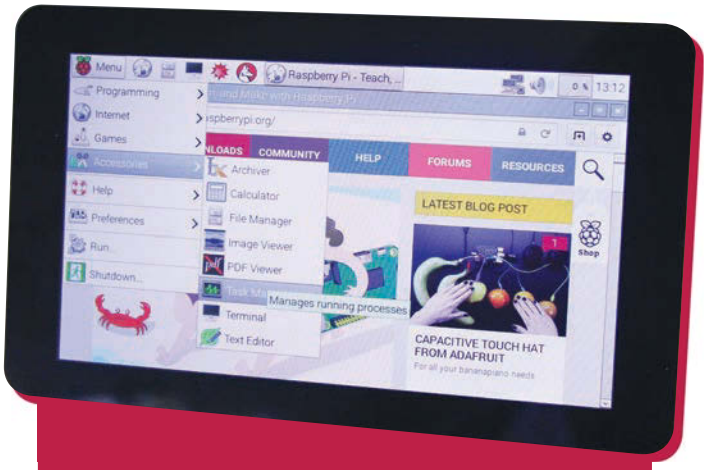
Analogue magazines and free history

I was really excited to see that you put all the original *The MagPi* issues up on your apps recently! I did enjoy them, and it was great to get all 30+ for free to join the rest of my digital collection of the mag. However, I do live in France and I'd love to get some of the print versions of the mag for my collection.

I've never seen any of the magazines on the shelf here, though – is there any way I can get analogue English copies of the magazine here?
Jay

Thanks, we were very proud of the fact we were able get all the magazines from before the relaunch up on our app! We're making sure they remain free forever on there as well, so that everyone can complete their collection without breaking the bank.

As for getting physical versions out to France, *The MagPi* is sold in the Raspberry Pi's Swag Store (**bit.ly/1ZX6yhV**) and ships to anywhere in the world. Including France! They're the same price as you'd buy them in the shop, although you'll have to pay a bit extra for shipping. Of course, if you really wanted to, you could grab the PDFs for the mags and print them yourselves. Ordering the issues might save you a bit of time, though.



An alternate display

I take my Raspberry Pi around with me when I travel, but I don't always have a way to display it on a proper monitor or TV screen to do any work on it. I do, however, take my laptop around with me wherever I travel. I've always wondered if there's a way for me to plug the Raspberry Pi into its HDMI port and view the picture on the screen – I've tried but never figured out if I was doing something wrong or had to switch 'input' or whatever.

Is there any way you could do a guide on how to get this to work?
Thanks,
Praveen

Unless your laptop has some very specific gear to do video capture or the like via HDMI ports, it's not going to be possible to hook the HDMI cable from the Raspberry Pi to your laptop and get it to display on there. The laptop's HDMI port will be an output, much like the Raspberry Pi's, so nothing will happen between the two.

That's not to say you can't view your Pi desktop on your laptop, though. The method most similar to what you want involves getting capture gear that can be plugged into your laptop via USB. You can plug the Raspberry Pi into there and then display it on the screen in a window – not ideal, but it will function as much the same as possible. Otherwise, you can use screen-sharing utilities that work with VNC to remotely dial into the Pi's screen from your laptop as long as they're on the same network. We'll try to get a tutorial like that sorted for the future!

FROM THE FORUM: SPACE IN YOUR HAND

Robot parts

I live in Germany and it's very hard to get the parts for the amazing little robot [from issue #38 of *The MagPi*]. Is it at all possible to get all of the robot bits from one supplier, or to get a kit instead? Some of the websites have sold out of some of the components like the line tracker already and I'd really like to build it! Is there any way you can help me out on this? Thanks,
Hermann Sailer

Unfortunately, in an attempt to keep the cost of the robot down so we could keep to the £50 limit we set ourselves, it was very difficult to have all the parts come from one supplier. The line tracker in particular was perfect for the size of the Pi Model A+ and required simpler wiring to get it to work. You may just have to be patient and wait for it to get back into stock!

As for a kit, though, there's actually a new cheap robot kit that has been put together as part of CamJam's Pi Wars this year, which you can get from the Pi Hut (**bit.ly/1hNITxX**). It's actually very similar to our robot in a lot of ways, and will cost you about the same as well. It's also selling out every now and then, so you may need to wait for it to be in stock!

If there's enough demand in terms of kits, though, we'd definitely look into it for the future.

The Raspberry Pi Forum is a hotbed of conversation and problem-solving for the community – join in via **raspberrypi.org/forums**

It's been mentioned that the Astro Pis that are going to the ISS are (a) Model B+, (b) running a special version of Raspbian Wheezy, and (c) have special cases.

The cases are, of course, really out of the question to be duplicated or sold for anything affordable by mere mortals. The Model B+ is easy to obtain, so no issues there. Would it be possible to set up a download of the specific version of Raspbian that is going to be flown so that those on the ground who wish to demo the Sense HAT and talk about Pis in space could run the 'proper' OS on them?
W H Heydt

We've been told by people in the know that the image used on the Astro Pi is not really special, other than carrying the competition code and an MCP (master control program) to select which code to run on it. They've also told us it's quite complex to set up at the moment. Either way, some of the competition code entries are up on GitHub, and more and more of the code for the Astro Pis will see a release over the next few months. So it sounds like you'll be able to get it eventually!

WRITE TO US

Have you got something you'd like to say?
Get in touch via **magpi@raspberrypi.org** or on The MagPi section of the forum at **raspberrypi.org/forums**

MATT RICHARDSON

Matt is Raspberry Pi's US-based product evangelist. Before that, he was co-author of *Getting Started with Raspberry Pi* and a contributing editor at *Make:* magazine.

GOING MAINSTREAM

Looking at the growth in our community, **Matt Richardson** thinks we're just getting started...

There's no doubt that ever since the first batch of Raspberry Pis started shipping, adoption of our favourite educational computer has been gaining momentum globally. The signals are out there. I'm sure many of you long-time enthusiasts can recall a moment when you've encountered Raspberry Pi in a context that you didn't quite expect. Perhaps on television, in a newspaper, or for sale in a store. Recently, a family member called to tell me that he was surprised to overhear someone talking about Raspberry Pi – and it wasn't me!

I've had a few of these moments recently. In June I was walking around Washington DC, proudly wearing my Raspberry Pi T-shirt. A total stranger passed me by, gave me a thumbs-up, and exclaimed, "Yeah, Raspberry Pi!" It took me completely by surprise. And after the launch of Raspberry Pi 2, my barber was asking about Pi because he had heard about it in the news.

What we're starting to see is Raspberry Pi entering the mainstream. I know that it feels like we're all part of a big movement right now, but I believe we're just on the brink of a really big breakthrough into being more recognisable and universal. It's bound to have a huge impact on our mission and our community.

For example, up until recently, buying a Raspberry Pi meant ordering online from a speciality store, one that provided hobby or industrial electronic components. If you managed to find a bricks-and-mortar store that sold Raspberry Pis, it probably also sold a wide variety of computer components. However, this is starting to change.

In the United States, Barnes & Noble Booksellers is now stocking Raspberry Pi kits and *The MagPi* in its 640 locations. In the UK, you can find Raspberry Pis in Currys, and you can buy *The MagPi* at WHSmith and Tesco. There's a chance that you picked up this copy of *The MagPi* at a place that also sells basic consumer products like groceries, books, or toys.

At outreach events, we now spend less time introducing people to Raspberry Pi and more time listening to what people are doing with their Raspberry Pis. When we do introduce people to Raspberry Pi, they often already have at least a vague notion of what we're all about, instead of being completely unfamiliar with it.

Pi for everyone

When we were younger, my brothers and I became fans of a particular band that wasn't mainstream and it felt like nobody had ever heard of them. (By the way, I won't name this band – it would be too embarrassing!) The first time we heard their music on the radio we were initially excited, but then became dismayed. This small band we liked was going mainstream and it wouldn't be as special to us anymore.

Why don't I feel this way as Raspberry Pi goes mainstream? It's because the bigger our community is, the better it makes Raspberry Pi. The more that people learn with it, the more they make with it, and the more they help others only improves Raspberry Pi as a resource for each of us.

There's plenty of Raspberry Pi to go around for everyone and I expect a broader, more novice group of users to join us. They're in luck; they have an incredibly welcoming community waiting for them. Those of you who have been with us since the early days: thank you. You're a huge part of how Raspberry Pi has been able to make it this far. And you can officially say that you liked Raspberry Pi before it was cool.

Expand your Pi

Stackable expansion boards for the Raspberry Pi

Serial Pi Plus

RS232 serial communication board. Control your Raspberry Pi over RS232 or connect to external serial accessories.

Breakout Pi Plus

The Breakout Pi Plus is a useful and versatile prototyping expansion board for the Raspberry Pi

ADC Differential Pi

8 channel 18 bit analogue to digital converter. I²C address selection allows you to add up to 32 analogue inputs to your Raspberry Pi.

IO Pi Plus

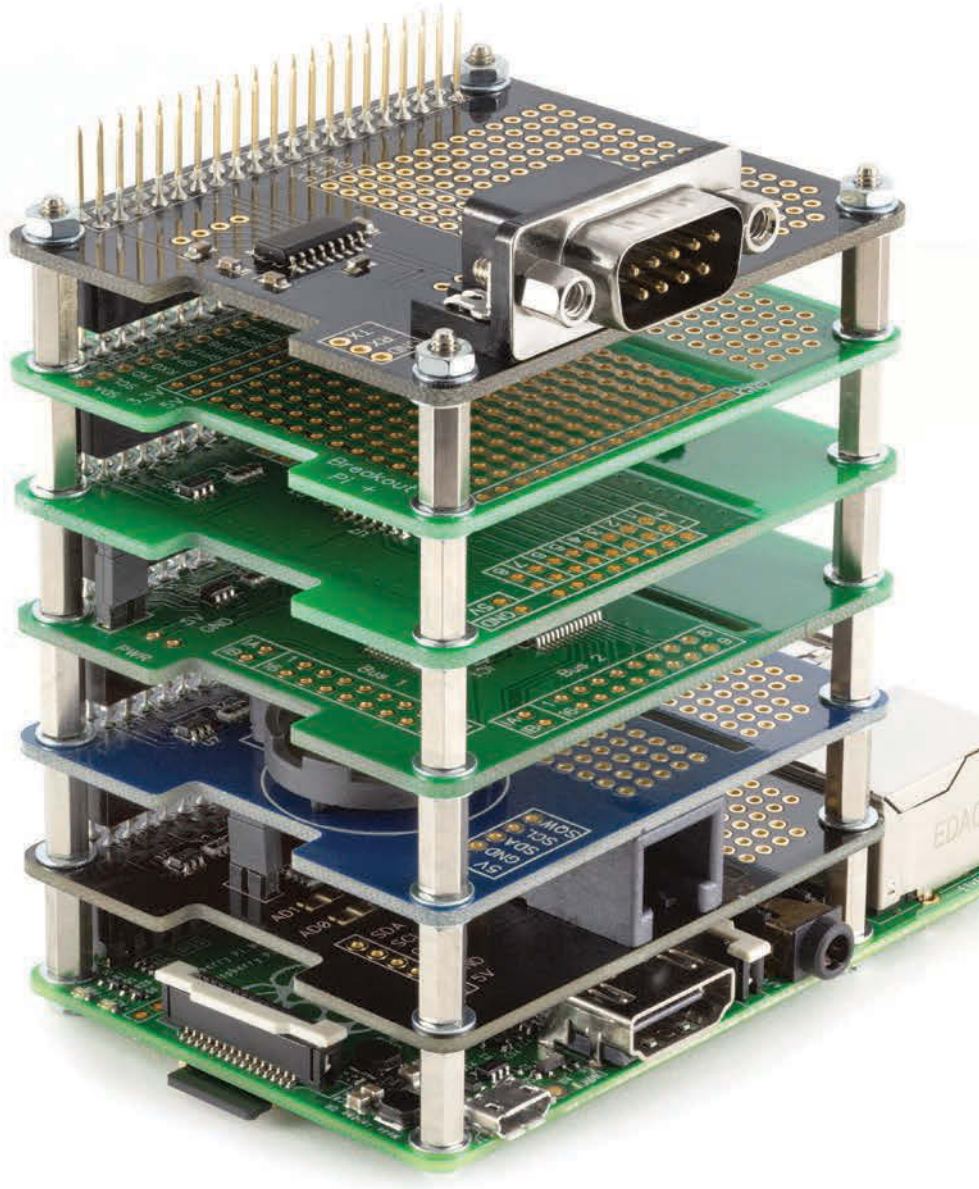
32 digital 5V inputs or outputs. I²C address selection allows you to stack up to 4 IO Pi Plus boards on your Raspberry Pi giving you 128 digital inputs or outputs.

RTC Pi Plus

Real-time clock with battery backup and 5V I²C level converter for adding external 5V I²C devices to your Raspberry Pi.

1 Wire Pi Plus

1-Wire® to I²C host interface with ESD protection diode and I²C address selection.



We also stock a wide range of expansion boards for the original Raspberry Pi models A and B

AB electronics UK

www.abelectronics.co.uk