FROM THE MAKERS OF The MagPi THE OFFICIAL RASPBERRY PI MAGAZINE

NEW 2022 UPDATE

RETRO GAMING WITH RASPBERRY PI

2ND EDITION

164 PAGES OF VIDEO GAME PROJECTS

BUILD AN **ARCADE MACHINE**

press start

VILROS

GET BACK IN TIME WITH RETRO GAMES

The 1980s was a golden era for video games. It was a vibrant, creative period where games were made by small teams and even individuals. Those early video game creators became famous, at least amongst the small crew of gamers eagerly following every new release.

Projects were personal and inspiration was taken from the everyday. There were games about regular jobs, like Trashman, or games set in developer's home towns, like Jack the Nipper. Not everybody had to be a Sabatour ninja or Turbo Lotus Esprit driver (although there were plenty of big dreams along the way).

This book enables you to rediscover the joy of retro gaming with Raspberry Pi, a small computer costing from just $35.

We show you how to transform this small computer into a games console, attach a controller, and get some classic games to play. With the right software, Raspberry Pi becomes a Mega Drive, Commodore Amiga, Sinclair Spectrum, or just about any other computer or console.

You can learn to make your own games, just like the classics you used to play. We'll show you how to make retro hits using a simple computer language called Python.

If you are up for a challenge, we will even show you how to build a full-sized arcade machine. Free to play forever.

We hope you enjoy this journey back in time.

Lucy Hattersley

FIND US ONLINE magpi.cc **GET IN TOUCH** magpi@raspberrypi.com

The MagPi

Available on the App Store

GET IT ON Google Play

EDITORIAL
Editor: **Lucy Hattersley**
Features Editor: **Rob Zwetsloot**
Contributors: **David Crookes, PJ Evans, Rosie Hattersley, Nicola King, Phil King, KG Orphanides, Mark Vanstone**

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

DESIGN
Critical Media: **criticalmedia.co.uk**
Head of Design: **Lee Allen**
Designers: **Sam Ribbits, Ty Logan**
Illustrator: **Sam Alder, Dan Malone**

MAGAZINE SUBSCRIPTIONS
Unit 6, The Enterprise Centre, Kelvin Lane, Manor Royal, Crawley, West Sussex, RH10 9PE | **+44 (0)207 429 4000**
magpi.cc/subscribe
magpi@subscriptionhelpline.co.uk

PUBLISHING
Publishing Director: **Russell Barnes**
russell@raspberrypi.com

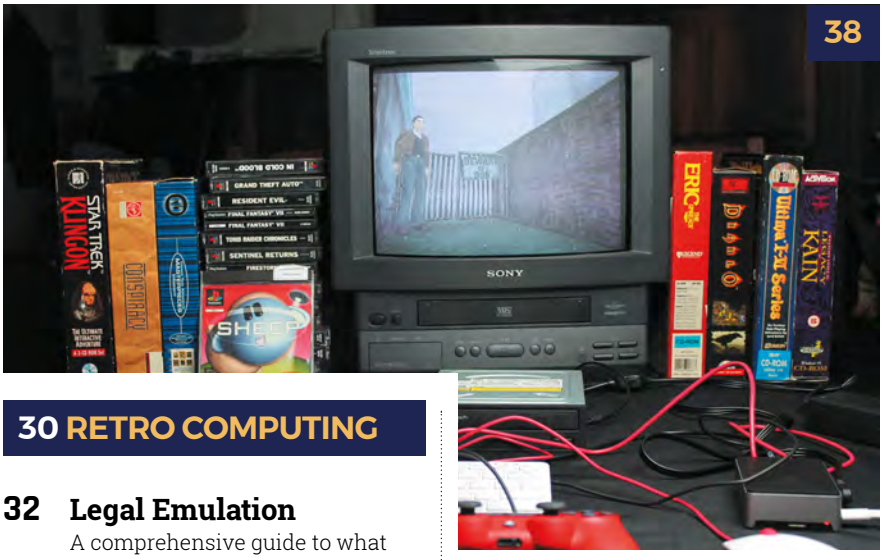
Advertising: **Charlotte Milligan**
charlotte.milligan@raspberrypi.com
Tel: +44 (0)7725 368887

Director of Communications: **Liz Upton**
CEO: **Eben Upton**

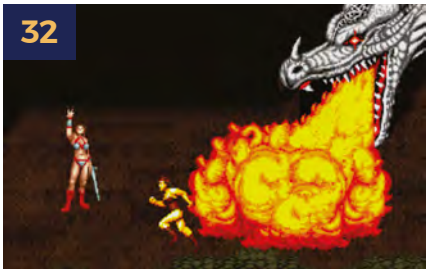
This official product is published by Raspberry Pi Ltd, Maurice Wilkes Building, Cambridge, CB4 0DS. The publisher, editor and contributors accept no responsibility in respect of any omissions or errors relating to goods, products or services referred to or advertised in the magazine. ISBN: 978-1-912047-77-2.

CONTENTS

28

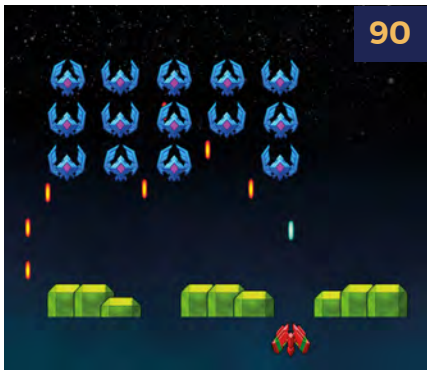


38


32


48

44

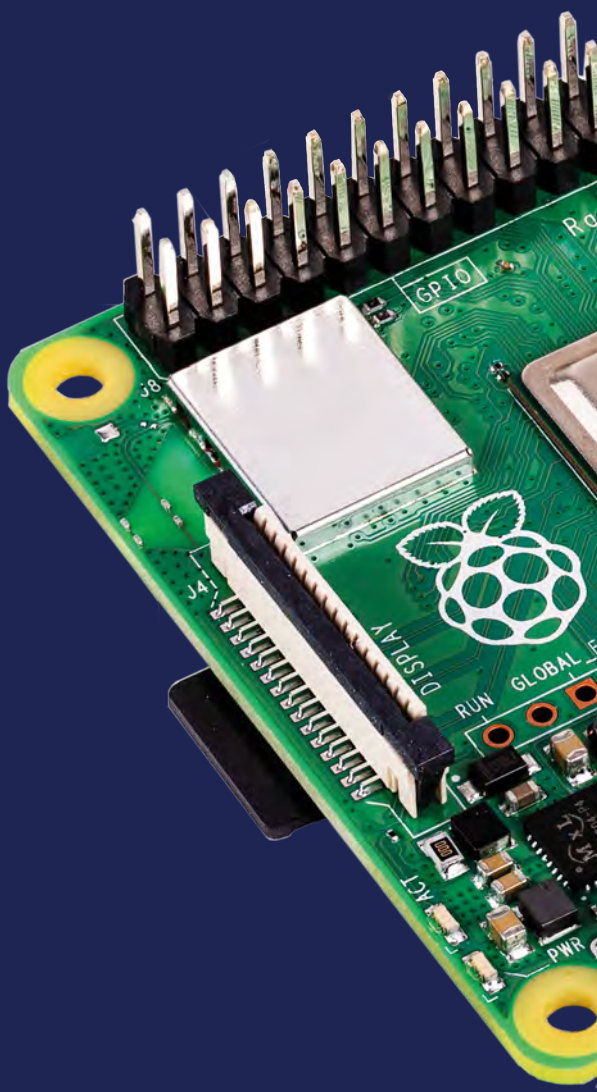


66 MAKE YOUR OWN GAMES

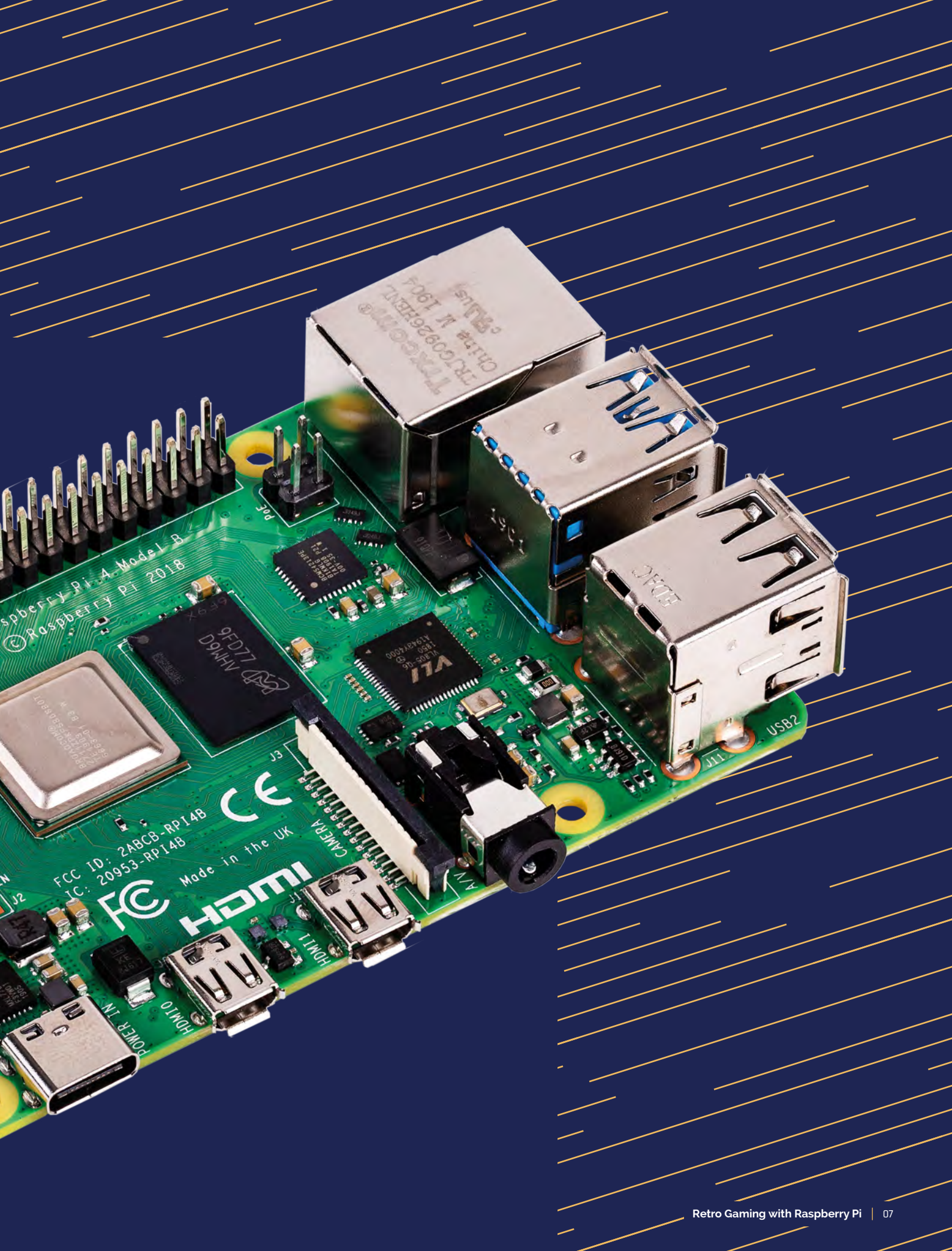
126 ARCADE PROJECTS

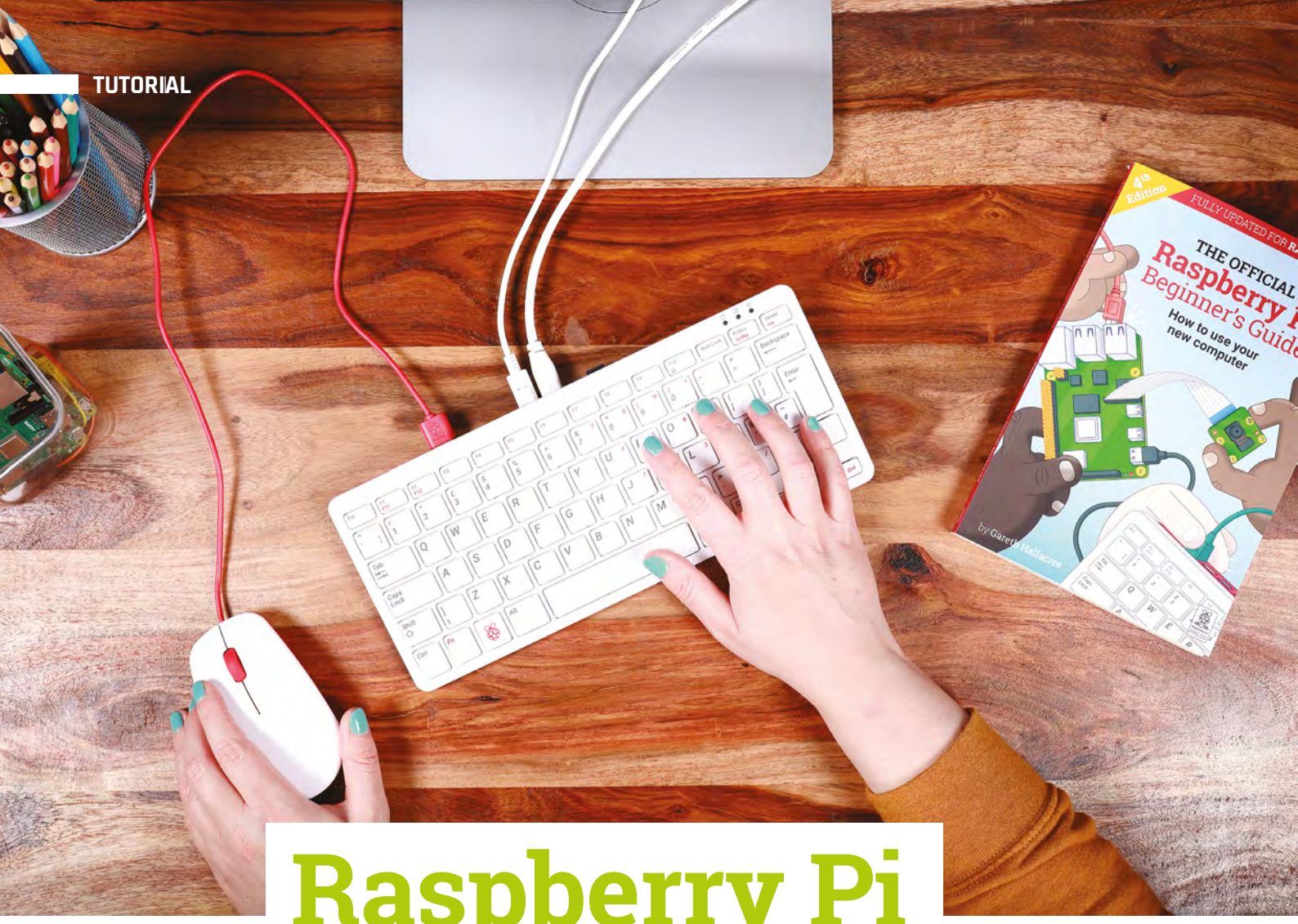
SET UP YOUR SYSTEM

EVERYTHING YOU NEED TO GET UP AND RUNNING



> Turning a Raspberry Pi device into a retro games console is a fun project


Raspberry Pi 4 Model B
©Raspberry Pi 2018
FCC ID: 2ABCB-RPI4B
IC: 20953-RPI4B
Made in the UK
HDMI
CAMERA
HDMI1
HDMI0
POWER IN
USB2
J3
J2
J11
POE



Raspberry Pi QuickStart Guide

Setting up Raspberry Pi is pretty straightforward. Just follow the advice of **Rosie Hattersley**

Congratulations on becoming a Raspberry Pi explorer. We're sure you'll enjoy discovering a whole new world of computing and the chance to handcraft your own games, control your own robots and machines, and share your experiences with other Raspberry Pi fans.

Getting started won't take long: just corral the extra bits and bobs you need on our checklist. Useful additions include some headphones or speakers if you're keen on using Raspberry Pi as a media centre, or a gamepad for use as a retro games console.

To get set up, simply use your pre-written microSD card (or use Raspberry Pi Imager to set up a card) and connect all the cables. This guide will lead you through each step. You'll find the Raspberry Pi OS, including coding programs and office software, all available to use. After that, the world of digital making with Raspberry Pi awaits you.

What you need

All the bits and bobs you need to set up a Raspberry Pi computer

A Raspberry Pi

Whether you choose the new Raspberry Pi 400; or a Raspberry Pi 4, 3B+, 3B; Raspberry Pi Zero, Zero 2 W (or an older model of Raspberry Pi), basic setup is the same. All Raspberry Pi computers run from a microSD card, use a USB power supply, and feature the same operating systems, programs, and games.

8GB microSD card

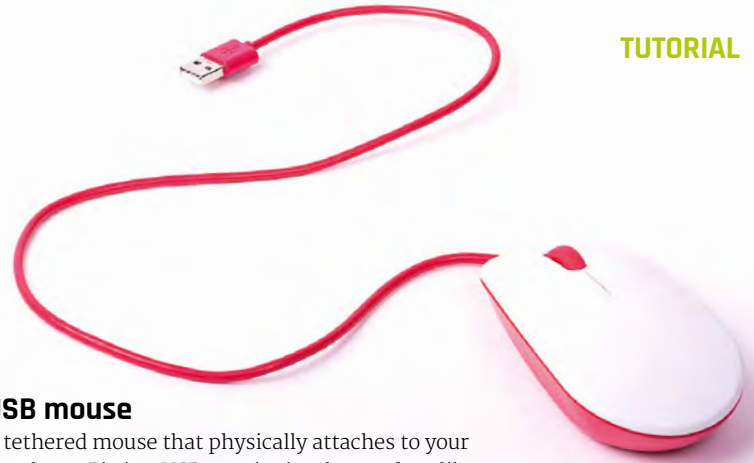
You'll need a microSD card with a capacity of 8GB or greater. Your Raspberry Pi uses it to store games, programs, and boot the operating system. Many Raspberry Pi computer kits come with a card pre-written with Raspberry Pi OS. If you want to reuse an old card, you'll need a card reader: either USB or a microSD to full-sized SD (pictured).

Windows/Linux PC or Mac computer

You'll need a computer to write Raspberry Pi OS to the microSD card. It doesn't matter what operating system this computer runs, because it's just for installing the OS using Raspberry Pi Imager (**magpi.cc/imager**).

USB keyboard

Like any computer, you need a means to enter web addresses, type commands, and otherwise control Raspberry Pi. The new Raspberry Pi 400 comes with its own keyboard. Raspberry Pi sells an official Keyboard and Hub (**magpi.cc/keyboard**) for other models.



USB mouse

A tethered mouse that physically attaches to your Raspberry Pi via a USB port is simplest and, unlike a Bluetooth version, is less likely to get lost just when you need it. Like the keyboard, we think it's best to perform the setup with a wired mouse. Raspberry Pi sells an Official Mouse (**magpi.cc/mouse**).

Power supply

Raspberry Pi 4 and Raspberry Pi 400 need a USB Type-C power supply. Raspberry Pi sells power supplies (**magpi.cc/usbcpower**), which provide a reliable source of power. Raspberry Pi 1, 2, 3, and Zero models need a micro USB power supply (**magpi.cc/universalpower**).

Display and HDMI cable

A standard PC monitor is ideal, as the screen will be large enough to read comfortably. It needs to have an HDMI connection, as that's what's fitted on your Raspberry Pi board. Raspberry Pi 4 and 400 can power two HDMI displays, but require a micro-HDMI to HDMI cable. Raspberry Pi 3B+ and 3A+ both use regular HDMI cables; Raspberry Pi Zero W needs a mini-HDMI to HDMI cable (or adapter).

USB hub

Raspberry Pi Zero and Model A boards have a single USB socket. To attach a keyboard and mouse (and other items), you should get a four-port USB hub (or use the official USB Keyboard and Hub which includes three ports). Instead of standard-size USB ports, Raspberry Pi Zero has a micro USB port (and usually comes with a micro USB to USB-A adapter).

Set up Raspberry Pi 400

Raspberry Pi 400 has its own keyboard – all you need to connect is the mouse and power

01 Connect a mouse

Connect a wired USB mouse to the white USB connection on the rear of Raspberry Pi 400. The two blue USB 3.0 connectors are faster and best reserved for external drives and other equipment that need the speed.

02 Attach the micro-HDMI cable

Next, connect a micro-HDMI cable to one of the micro-HDMI sockets on the rear of Raspberry Pi 400. The one next to the microSD card slot is the first one, but either connection should work. Connect the other end of the HDMI cable to an HDMI monitor or television.

03 The microSD

If you purchased a Raspberry Pi 400 Personal Computer Kit, the microSD card will come with Raspberry Pi OS pre-installed. All you need to do is connect the power and follow the welcome instructions. If you do not have a Raspberry Pi OS pre-installed microSD card, follow the instructions later in 'Set up the software'.

Set up Raspberry Pi

Raspberry Pi 4 / 3B+ / 3 has plenty of connections, making it easy to set up

01 Hook up the keyboard

Connect a regular wired PC (or Mac) keyboard to one of the four larger USB-A sockets on a Raspberry Pi 4 / 3B+/ 3. It doesn't matter which USB-A socket you connect it to. It is possible to connect a Bluetooth keyboard, but it's much better to use a wired keyboard to start with.

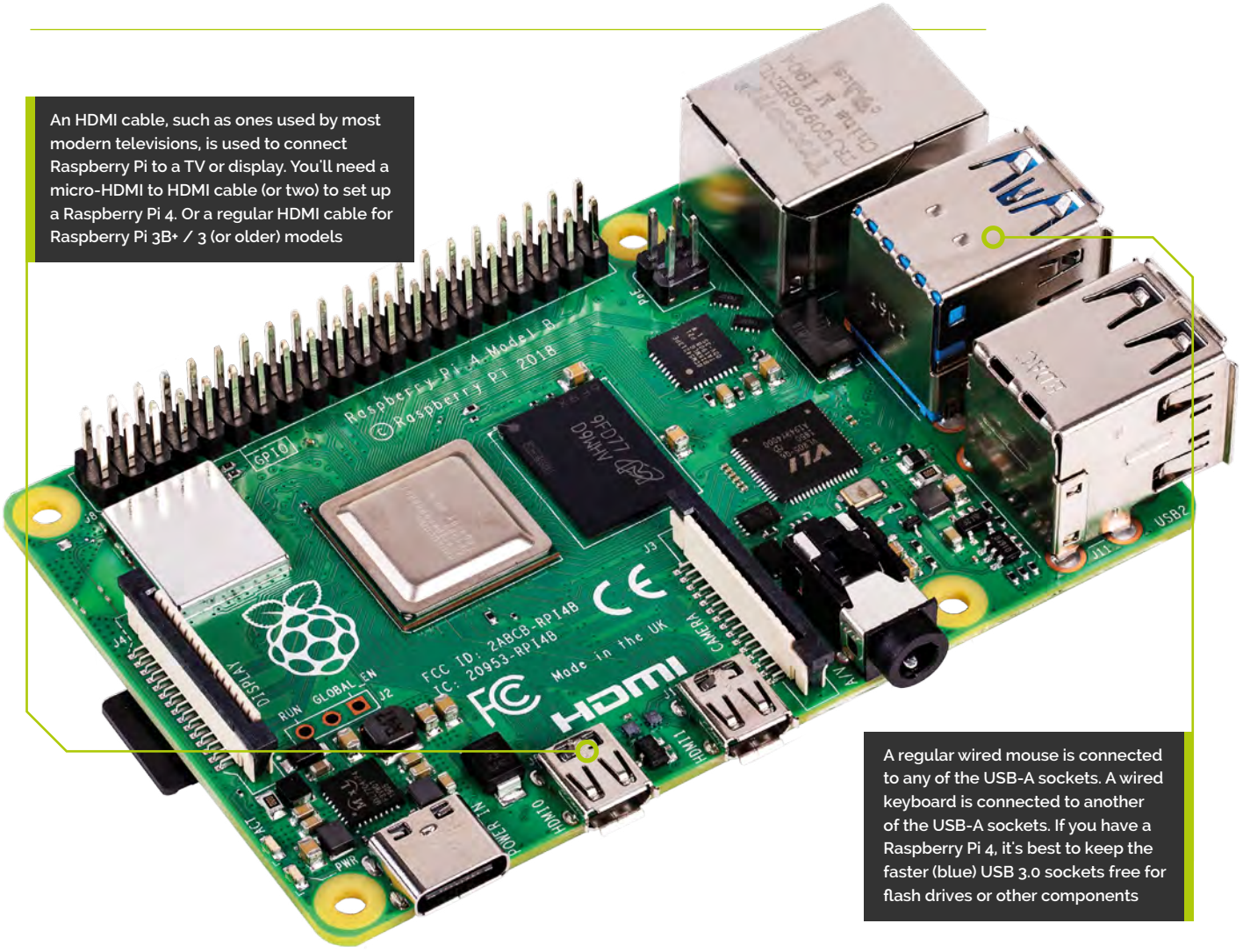
02 Connect a mouse

Connect a USB wired mouse to one of the other larger USB-A sockets on Raspberry Pi. As with the keyboard, it is possible to use a Bluetooth wireless mouse, but setup is much easier with a wired connection.

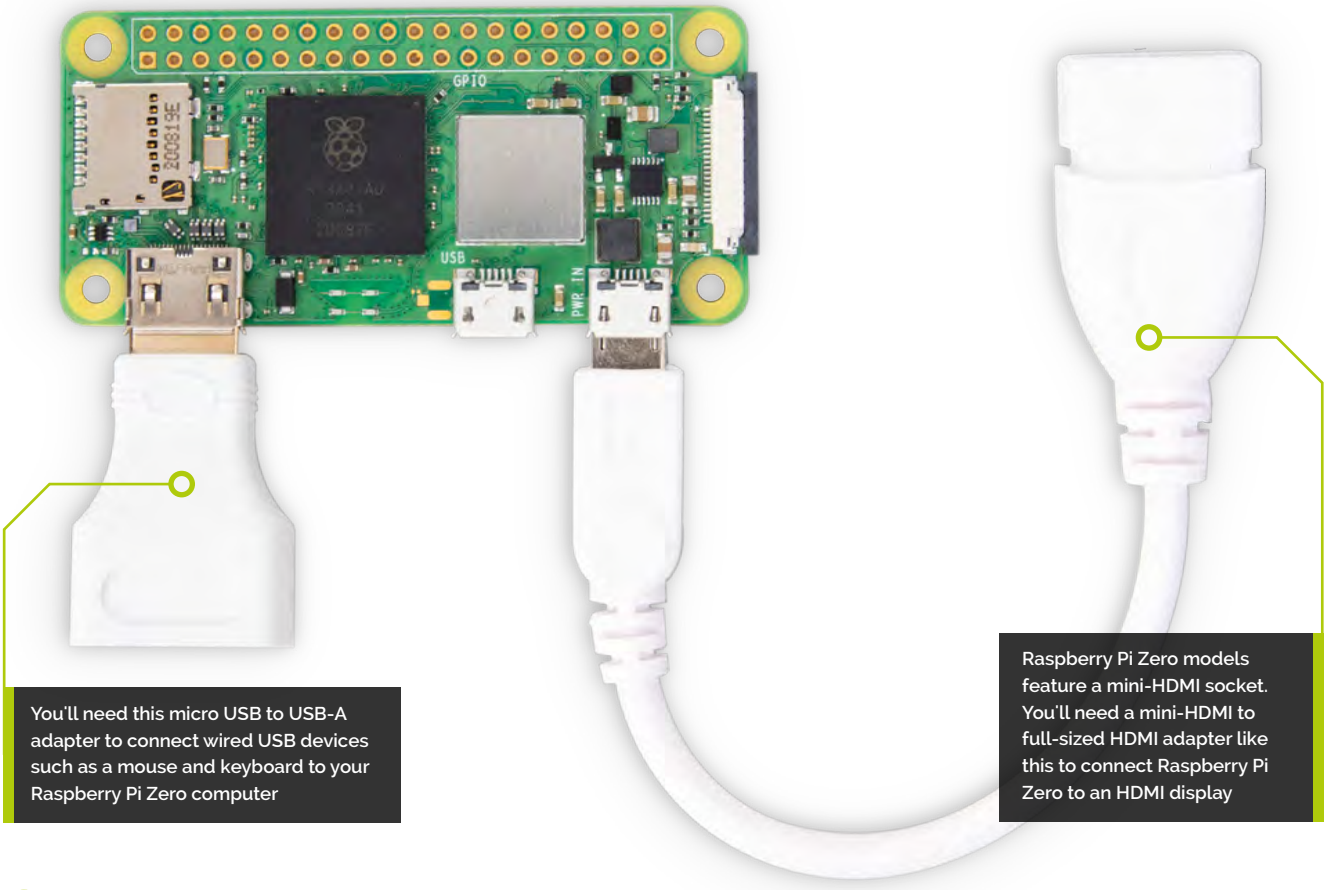
03 HDMI cable

Next, connect Raspberry Pi to your display using an HDMI cable. This will connect to one of the micro-HDMI sockets on the side of a Raspberry Pi 4, or full-size HDMI socket on a Raspberry Pi 3/3B+. Connect the other end of the HDMI cable to an HDMI monitor or television.

An HDMI cable, such as ones used by most modern televisions, is used to connect Raspberry Pi to a TV or display. You'll need a micro-HDMI to HDMI cable (or two) to set up a Raspberry Pi 4. Or a regular HDMI cable for Raspberry Pi 3B+ / 3 (or older) models



A regular wired mouse is connected to any of the USB-A sockets. A wired keyboard is connected to another of the USB-A sockets. If you have a Raspberry Pi 4, it's best to keep the faster (blue) USB 3.0 sockets free for flash drives or other components



Set up Raspberry Pi Zero

You'll need a couple of adapters to set up a Raspberry Pi Zero / Zero 2 W

01 Get it connected

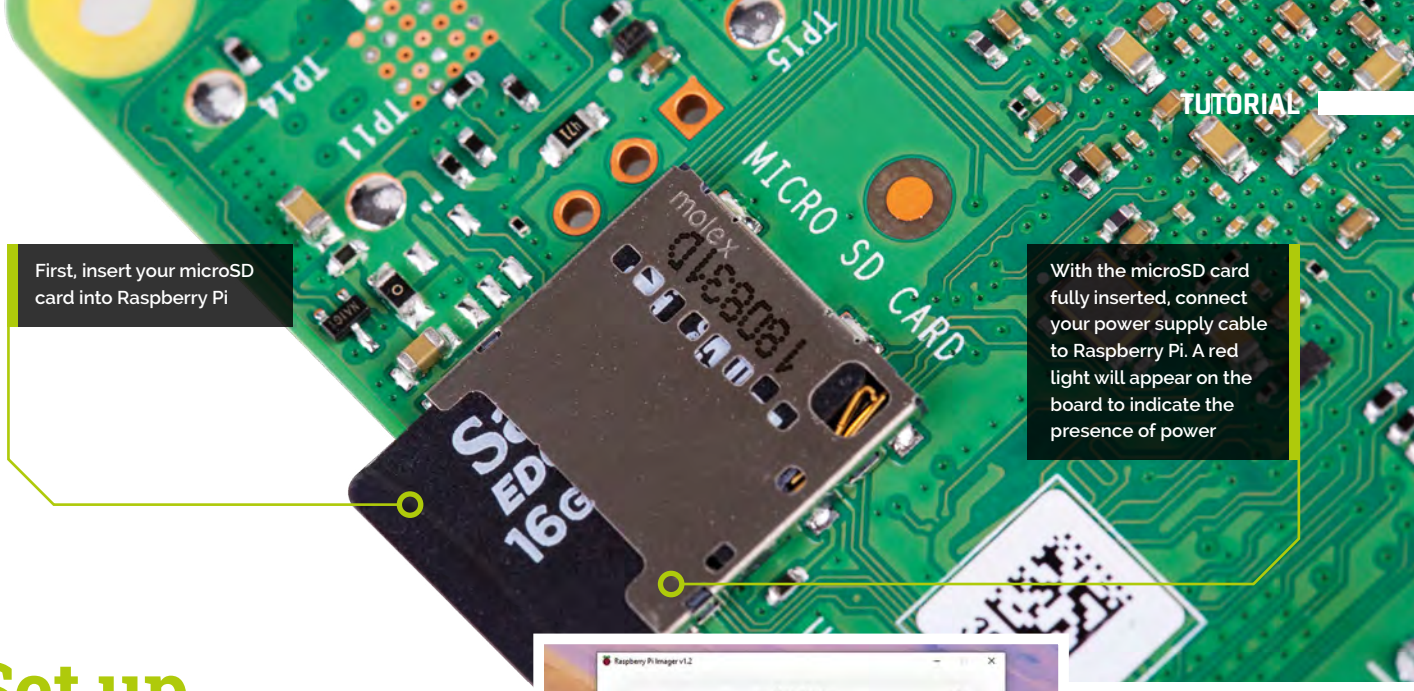
If you're setting up a smaller Raspberry Pi Zero or new Zero 2 W, you'll need to use a micro USB to USB-A adapter cable to connect the keyboard to the single micro USB connection on a Raspberry Pi Zero. The latter model has only a single micro USB port, which means you'll need to get a small USB hub or use an all-in-one mouse and keyboard with your Raspberry Pi Zero.

02 Mouse and keyboard

You can either connect your mouse to a USB socket on your keyboard (if one is available), then connect the keyboard to the micro USB socket (via the micro USB to USB-A adapter). Or, you can attach a USB hub to the micro USB to USB-A adapter.

03 More connections

Now connect your full-sized HDMI cable to the mini-HDMI to HDMI adapter, and plug the adapter into the mini-HDMI port in the middle of your Raspberry Pi Zero W. Connect the other end of the HDMI cable to an HDMI monitor or television.



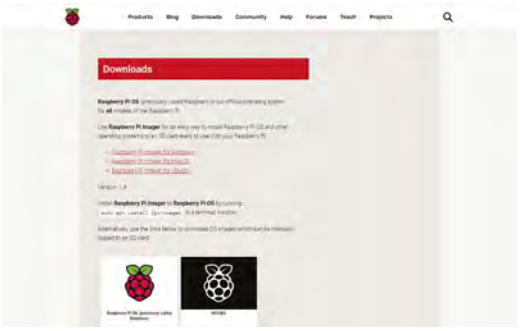
First, insert your microSD card into Raspberry Pi

With the microSD card fully inserted, connect your power supply cable to Raspberry Pi. A red light will appear on the board to indicate the presence of power

Set up the software

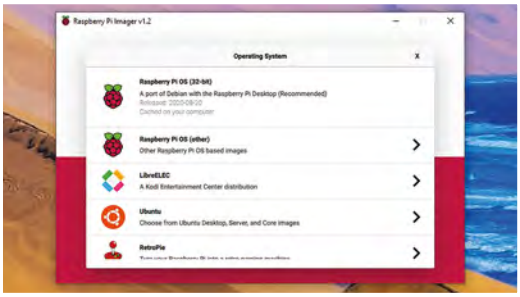
Use Imager to install Raspberry Pi OS on your microSD card and start your Raspberry Pi

Now you've got all the pieces together, it's time to install an operating system on your Raspberry Pi so you can start using it. Raspberry Pi OS is the official software for Raspberry Pi, and the easiest way to set it up on your Raspberry Pi is to use Raspberry Pi Imager. See the 'You'll Need' box and get your kit together.



01 Download Imager

Raspberry Pi Imager is available for Windows, Mac, and Linux computers. You can also install it on Raspberry Pi computers, to make more microSD cards once you are up-and-running. Open a web browser on your computer and visit **magpi.cc/imager**. Once installed, open Imager and plug your microSD card into your computer.



02 Choose OS

Click on 'Choose OS' in Raspberry Pi Imager and select Raspberry Pi OS (32-bit). Click 'Choose SD card' and select the microSD card you just inserted (it should say 8GB or the size of the card next to it). Click on 'Write'. Your computer will take a few minutes to download the Raspberry Pi OS files, copy them to the microSD card, and verify that the data has been copied correctly.

03 Set up Raspberry Pi

When Raspberry Pi Imager has finished verifying the software, you will get a notification window. Remove the microSD card and put it in your Raspberry Pi. Plug in your Raspberry Pi power supply and, after a few seconds, a blue screen will appear with 'Resizing Filesystem'. It will quickly vanish and be replaced by 'Welcome to Raspberry Pi'. Click on Next and follow the on-screen instructions to set up Raspberry Pi OS and start using your new computer.

Top Tip

Is your card ready?

You don't need to do this if your Raspberry Pi came with a card pre-written with Raspberry Pi OS.

You'll Need

- A Windows/Linux PC or Apple Mac computer
- A microSD card (8GB or larger)
- A microSD to USB adapter (or a microSD to SD adapter and SD card slot on your computer)
- Raspberry Pi Imager **magpi.cc/imager**

Set up RetroPie

Build a retro gaming box and install RetroPie

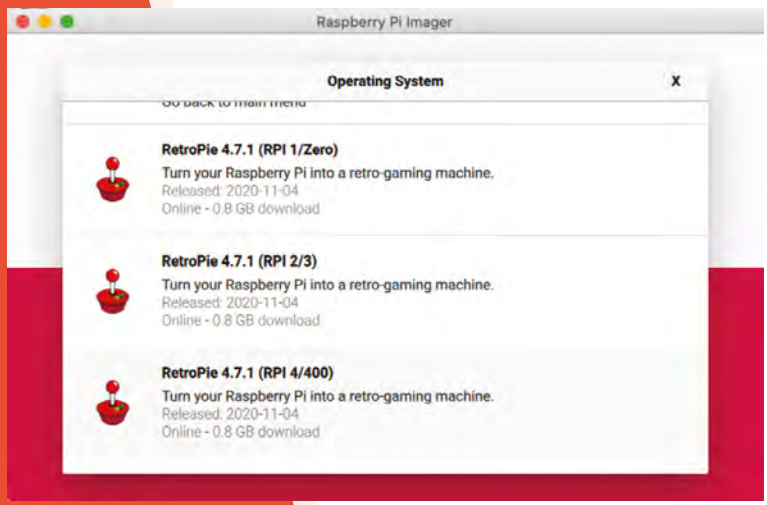
PUT IT TOGETHER

01 Install RetroPie

RetroPie allows you to emulate several game systems. We advise using Raspberry Pi Imager (**magpi.cc/imager**) to put RetroPie onto your microSD card. Attach your microSD card to your computer and open Imager. Click 'Choose OS', select 'Emulation and game OS' then 'RetroPie'; choose the version that matches your Raspberry Pi model: 'RPI 1/Zero', 'RPI 2/3' or 'RPI 4/400'. Click 'Choose Storage', select your microSD card. Finally, click 'Write'.

02 Install Raspberry Pi

Before setting up RetroPie, it's a good idea to install your Raspberry Pi into your selected case. We like the Argon One M.2 for a few reasons – it's secure, it keeps your Raspberry Pi cool, it can use a Nanosound DAC if that's your thing, and it routes all the inputs to the rear of the case. Because it has all these bells and whistles, it does take an extra step to install. The instruction manual that comes with the Argon One shows you how to add the daughterboard to Raspberry Pi. Follow the instructions supplied with your case.



▲ Raspberry Pi Imager makes it easy to install RetroPie – choose the RPI 4/400 version

03 Hook it up

For first-time setup, we recommend connecting a USB keyboard, along with having it connected to a monitor instead of a TV for comfort – however, connecting it to your selected TV is also fine. If you also plan to use a wired game controller, connecting it now is also a good idea. The last thing to connect should be the power supply.

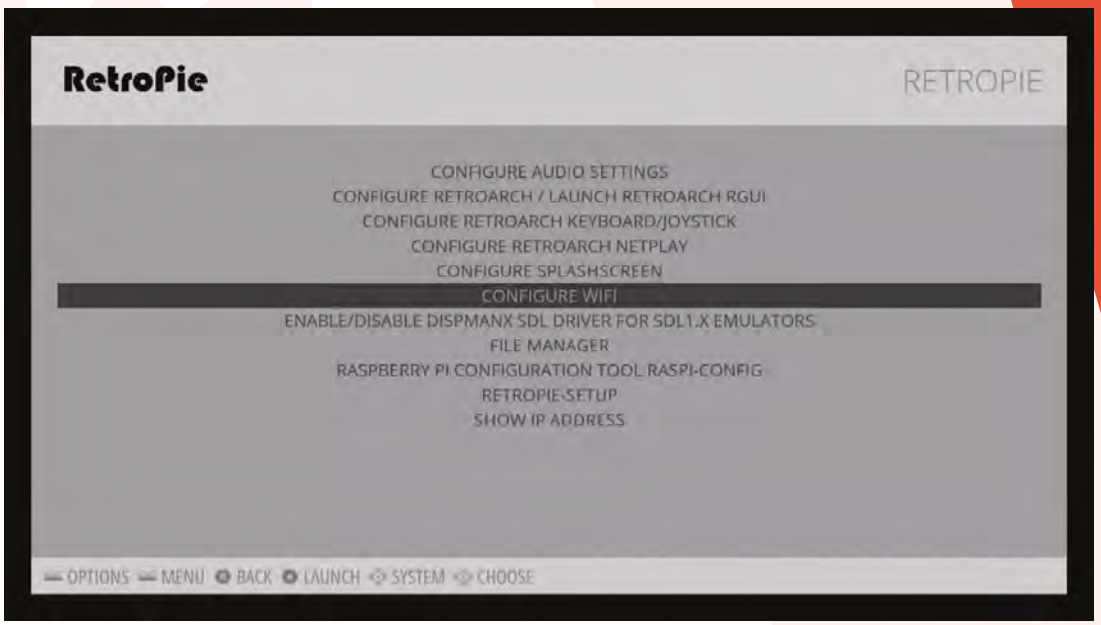
If you want to add an M.2 SSD drive for increased internal storage, do so now!

INITIAL SETUP

01 First boot

With a power supply plugged in and switched on, press the power button on the back of the case to boot up your Raspberry Pi. RetroPie will do some initial automated setup stuff, then ask you to configure your games controller

▲ Press the relevant buttons to configure your games controller; hold any button to skip a function



▲ Select the Configure WiFi option from the RetroPie menu and enter your network details

buttons. If you don't have a controller connected yet, you can press **F4** on the keyboard to get to the main menu. If you've run out of buttons to assign, hold down any button and it will skip the function. Keep doing this until you get to the end, and confirm 'OK' to finish.

If you're using a wired network and controllers, you can stop here! Just add ROMs over the network or via a USB stick to play games!

> Make sure you turn the controller on just after the reboot

02 Wireless connection

If you don't intend to use an Ethernet connection, you can connect to your wireless network in the RetroPie menu. Select 'Configure WiFi' and it will open a text menu. Select 'Connect to WiFi' and choose your network from the list. Enter your password and hit OK – you may need to wait a moment or two, but it should then be fully connected.

To transfer ROMs over the network, go to **\\RETROPIE** in the Windows File Explorer, or **smb://retropie** on Mac, and open up the **roms** folder

Stream from Steam

RetroPie allows you to use Steam Link software to stream games from a gaming PC straight to your TV! To do this, go to the RetroPie menu and then RetroPie Setup. Go down to Manage Packages and choose Experimental Packages from it. In that list will be 'steamlink' – install it and it will then appear in the main games menu, ready for you to start streaming from a gaming PC running Steam with Remote Play on.

03 Bluetooth controllers

Press **F4** and you'll open the command line. From there, make sure Bluetooth is installed using the command:

```
sudo apt install pi-bluetooth
```

Type **exit** to return to the graphical interface. Go to the menu and select Bluetooth Configuration. Select Register and Connect Bluetooth Devices while your Bluetooth controller is in pairing mode, then select it from the screen and follow the pairing instructions. Once connected, you may need to reboot your Raspberry Pi before configuring the buttons – make sure you turn the controller on just after the reboot.

For troubleshooting and tips on specific controllers, especially for Sony, Microsoft, and Nintendo consoles, look at the docs: **retropie.org.uk/docs**

Install Argon ONE software

To install the fan control to the system, press **F4** to get to the command line, and enter the following:

```
curl https://download.argon40.com/argon1.sh | bash
```

SUBSCRIBE TODAY FROM ONLY £5

SAVE UP TO 35%

Subscriber Benefits

- **FREE Delivery**
  Get it fast and for FREE
- **Exclusive Offers**
  Great gifts, offers, and discounts
- **Great Savings**
  Save up to 35% compared to stores

Rolling Monthly Subscription

- Low monthly cost (from £5)
- Cancel at any time
- Free delivery to your door
- Available worldwide

Subscribe for 12 Months

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

Free Raspberry Pi Zero 2 W with 12 Month upfront subscription only (no Raspberry Pi Zero 2 W with Rolling Monthly Subscription)

Subscribe by phone: **01293 312193**

Subscribe online: **magpi.cc/subscribe**

Email: **magpi@subscriptionhelpline.co.uk**

RETRO GAMING HARDWARE

REVIEWS OF THE TOP KIT FOR RETRO GAMERS

> Retro gaming consoles need a controller – arcade games are little fun to play with a keyboard and mouse


DELETE
ENTER
CAPS SHIFT
SYMBOL SHIFT
JOY 2

PiBoy DMG

SPECS

CONTROLS:
Ten buttons, eight-way D-pad, analogue joystick

POWER:
4500 mAh LiPo battery

DISPLAY:
3.5-inch LCD DPI 640×480

EXTRAS:
On/off button, HDMI out

▶ Experimental Pi ▶ **magpi.cc/piboydmg** ▶ £90 / $120

Is this seemingly ultimate handheld worth the price? **Rob Zwetsloot** investigates

A Raspberry Pi retro gaming case based around a classic handheld design is not exactly uncommon, and you might have turned the page here and been a bit sceptical. We were too when it landed on our desk; however, we were pleasantly surprised because that's the only feature of the PiBoy DMG that it shares with other similar devices.

▶ The PiBoy DMG takes its design cues from classic handheld consoles

Even on paper it has some interesting features – sure, it has a ton of buttons and a battery etc. (read the specs box for all that), but it also has an active cooling fan, an analogue joystick, and even a brightness control wheel for the screen – something very reminiscent of the contrast control on the original handheld console.

Usually, a lot of these kits can feel very cheap and rough, using standard 3D-printed parts for everything that can feel uncomfortable and flimsy and don't really have the nicest aesthetic. The PiBoy feels more like the real deal: the main case is sturdy, the buttons are nice to use, and even the analogue stick has a little click-down thing. Unfortunately, like a lot of original form factor builds, the 'shoulder' buttons on the rear are a bit fiddly. With six face buttons, though, you're probably set for playing any games up until the 16-bit era.

Pocket emulation

Speaking of playing games, the software on the PiBoy is a slightly modified version of RetroPie,

▲ If you really fancy connecting a portable console to Ethernet, you can do that!

Different, simple adapters need to be used depending on which Raspberry Pi you install

with specific Experimental Pi splash screens and branding to the startup. Thanks to this, you're only really limited by your Raspberry Pi choice, with Raspberry Pi Zero, Raspberry Pi 3/3B+, and Raspberry Pi 4 supported.

Because of this, the kind of games you'd be running on RetroPie systems run as smoothly as you'd expect. The LCD screen outputs at a fairly reduced resolution anyway, which reduces some of the load. With the fan on the rear of the PiBoy, we didn't find it getting too hot with a Raspberry Pi 4 in it, although the whining of the fan is slightly unnerving for a handheld and sounds like a CD. You can play the PiBoy in any position you wish without scratching anything, thankfully.

> The LCD screen outputs at a fairly reduced resolution

The various adapters and such for the PiBoy allow for all the output and input options of the installed Raspberry Pi to be accessible. As well as USB sticks which can be used for storage, and easy access to the microSD card, you can even plug in headphones and use a (regular size) HDMI cable to plug it into your TV. Use the available USB ports for some USB controllers and you have a very portable plug-and-play box.

Amazingly, it also has a special Steam Link function. You'll likely be connected to wireless LAN on the PiBoy and if you have a decent connection, it's amazing to play some games in your hands in your own home.

It's a pretty fantastic piece of kit, and we think it earns its price tag. Just don't rely on the shoulder buttons.

The battery case is the same as the original, albeit this one uses a much more powerful battery

Verdict

An incredible portable retro gaming build, this has just about everything you'd want from a Raspberry Pi-based handheld console kit.

10/10

PicoSystem

SPECS

INTERNALS:
RP2020 chip (dual Arm Cortex-M0+ running at up to 133MHz with 264kB of SRAM), 16MB of QSPI flash supporting XiP

DISPLAY:
1.54" colour SPI IPS LCD (240×240 pixels)

FEATURES:
D-pad and buttons, 525 mAh LiPo battery, piezo buzzer/speaker, CNC-milled aluminium case, wrist strap

▶ Pimoroni ▶ **magpi.cc/picosystem** ▶ £59 / $66

A handheld console built around RP2040 and with game development in mind. **Rob Zwetsloot** boots it up.

Extremely tiny handheld consoles have not always had the best track record in the microcomputer space. A few Raspberry Pi Zero efforts ended up being just a little too small and finnicky for actual use, and it didn't quite help that they often used hard 3D-printed buttons that were uncomfortable. We're pleased to say that Pimoroni's PicoSystem, while still small, manages to avoid this.

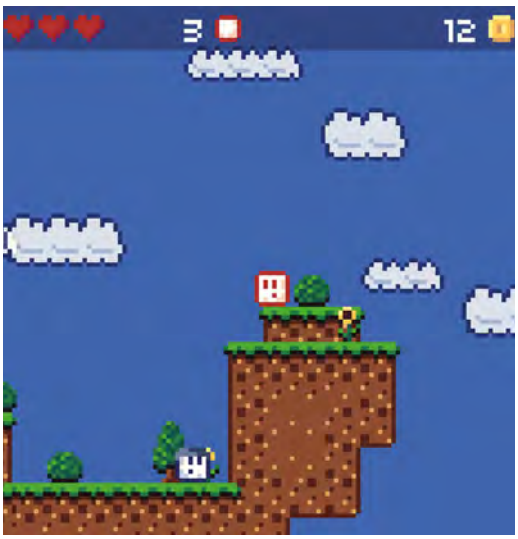
It's made from milled aluminium, has proper buttons, and a nice little square screen with 240×240 pixels. It feels nice to hold – there's a bit of heft – and holding it is not uncomfortable. It comes with a game pre-installed: Super Square Bros, a platformer. However, the main draw really is that you can make games for it yourself.

Make your own fun

PicoSystem uses its own official API, which works MicroPython and C++ – just like a standard Raspberry Pi Pico or other RP2040-based systems, allowing you to easily transfer skills over from elsewhere. CircuitPython, which is based on MicroPython is supported as well, and there's even a 32blit SDK, allowing you to port over games from Pimoroni's 32blit handheld console.

The standard C++ and MicroPython API adds loads of functions to make creating games slightly easier, including camera control, linking to buttons, and primitives for drawing sprites more easily than building them pixel by pixel. There are different in-engine effects you can apply to text and such – it's a meaty API that you can get a lot out of for just a simple pixel game on a limited piece of hardware.

The current games available for PicoSystem run absolutely fine as well, although the lack of a proper speaker is a little noticeable. The piezo buzzer sounds are quite charming in their own way, though, and smart use of it can create some nice retro bleeps and bloops.



▶ There are already some fun games that have been made for it

> The main draw really is that you can make games for it yourself

Pocket games

Switching out games is a little more tricky than changing cartridges, though – there's limited space on the PicoSystem, and you need to connect it to a computer to do a quick re-flashing to play a different game. It's not too frustrating, as it's pretty quick, so for development you don't have to wait too long. If you're out and about with it hanging from your wrist, though (it comes with a cool lanyard), you will be fairly limited.

Yet that hasn't really stopped us. It's lovely to take around with you, the battery lasts for ages and charges pretty quickly, and you can make some really beautiful stuff for it. We look forward to see what kind of games people make for it.

▲ The PicoSystem is beautifully designed and feels premium

◀ It has a charm loop holder bit which is extremely important

Verdict

An incredibly cool, tiny handheld that you can fairly easily develop games for. We just wish it had more storage.

9/10

Deluxe Arcade Controller Kit

▶ Monster Joysticks ▶ **monsterjoysticks.com** ▶ £100 / $127

Rob Zwetsloot builds a mini arcade machine with this all-in-one controller kit from Monster Joysticks

On page 134 of this book, we take you through a comprehensive arcade machine build, including a complete wooden build of the cabinet itself. While it's certainly impressive, not everyone has the space, time, or money for one. This is where awesome little kits like this one from Monster Joysticks come in.

You've probably seen this type of kit before – it's an all-in-one arcade joystick and case for your Raspberry Pi that turns it into a small and portable arcade machine. Just hook it up to the nearest television and you're ready for some serious retro gaming action.

Unlike the stocking filler plug-and-play consoles, this kit requires you to build your gaming system and supply a Raspberry Pi board to power it. Construction is very simple, though: there are six acrylic panels for each side of the box and only eight screws required to fasten them all together.

Quality components

The kit comes with nine genuine Sanwa arcade buttons and a Sanwa joystick, which just simply click into the acrylic panels as you build them.

To wire up the buttons and joystick, a little add-on board is provided with colour-coded wires. They can be a little tricky to properly attach to the connections as the connectors themselves are a bit tight, but you don't have to worry too much about wires getting tangled up. You may also need to push down the top panel a bit due to resistance of all the wires, but otherwise it all fits fairly neatly inside. You can find the full build instructions on the Monster Joysticks website: **magpi.cc/2i3iQp8**.

The build took us just shy of three episodes of *The Simpsons*, so make sure you set aside about an hour for the job. Our only real complaint about the build is that, while all the ports and even microSD card slot are readily accessible, your Raspberry Pi can only be removed by taking the case apart. It

will only take a couple of minutes to remove it, but we'd have preferred it to be a little easier.

The final part of the build involves attaching little rubber feet to the bottom – very welcome, as the case had been slipping a bit on the glass table it had been built on.

The stick feels solid and has a decent weight to it thanks to the included components, so you feel pretty safe giving the buttons and joystick a proper workout. The included Sanwa components are quite important as not only are they high-quality and can survive a bit of classic button mashing/frame-perfect combo-timing, they're also quite customisable. For instance, if you don't fancy the button colour scheme, you can always swap them out. The joystick itself can also be customised: the version that comes with the kit has square four-way gates, but they can be upgraded to an octagonal eight-way gate, or any other gate style if you prefer.

Quick configuration

Software customisation for RetroPie is also very simple. With a custom add-on board to connect the controls to Raspberry Pi over GPIO, we initially feared we'd have to download custom scripts for the job. Not so, though, and while you do need to go into the RetroPie configuration menu and install an extra driver, it's all quick and included in the RetroPie archive. Once that's done, you can configure the stick controls, as well as any extra controllers you've plugged into the USB ports.

> "The stick feels solid and has a decent weight to it thanks to the included components"

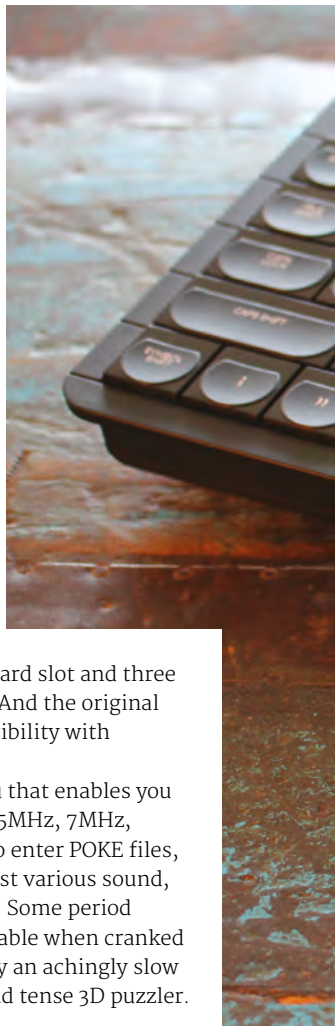
This kit is a great, solid package and it looks good as well. We recommend investing in some nice, long HDMI and USB cables to power the box and don't be afraid to put some sticks or a little custom decal onto the case as well. With Christmas coming up, it may just be the perfect gift for someone.

Verdict

A great little kit. It's a fun build but also a good quality product to use. We'd prefer our Raspberry Pi to be a bit more accessible, but otherwise the high customisability is a big plus.

8/10

ZX Spectrum Next Accelerated



SPECS

PROCESSOR: Z80 on Xilinx Spartan-6 XC6SLX16 FPGA

MEMORY: 1MB RAM (expandable to 2MB internally)

AUDIO: 3 × AY-3-8912 audio chips with stereo output

WIRELESS: ESP8266 WiFi module

ACCELERATOR BOARD: Raspberry Pi Zero W

▶ SpecNext ▶ **specnext.com** ▶ £230 / $288

Raspberry Pi Zero adds audio and realistic loading to this reimagined ZX Spectrum computer. By **Lucy Hattersley**

Following several years of development, The ZX Spectrum Next Accelerated – a reimagining of the popular 1980s computer – has finally arrived. Part of the Spectrum's modernisation is a Raspberry Pi Zero acting as an accelerator, melding the power of a 1980s classic with the greatest computer of the modern age.

Inside the box you'll find the ZX Spectrum Next computer, a power supply, and a spiral-bound user manual which is pleasingly similar to the original but packed with information from BASIC programming to machine code.

The keyboard is a thing of beauty. The keys are responsive, although the layout is a bit weird after years of muscle memory bonded to PC.

It's packed with connections: HDMI and VGA for video out; 3.5mm ear and mic mini-jacks; PS/2 for keyboard and mouse; plus the mini HDMI and micro USB ports of Raspberry Pi Zero; and two 9B9 joystick ports (compatible with Kempston, Cursor, and ZX Interface 2 Protocols). To the left of the device sits a full-size SD card slot and three buttons: Reset, Drive, and NMI. And the original Expansion port provides compatibility with classic hardware.

The NMI button opens a menu that enables you to flick between turbo modes: 3.5MHz, 7MHz, 14MHz, and 28MHz. You can also enter POKE files, browse memory banks, and adjust various sound, graphical, and memory settings. Some period games become wonderfully playable when cranked up to 28MHz: Sentinel, originally an achingly slow trudge, becomes a fast-paced and tense 3D puzzler.

Z80 and beyond

The heart of the Spectrum Next is a Xilinx Spartan-6 XC6SLX16 FPGA (field-programmable gate array, **magpi.cc/spartan6**). FPGA isn't

▼ A range of modern and classic ports make for a versatile computer

▲ The keyboard, designed by Rick Dickinson, has come together perfectly

▲ Its classic design is very similar to that of the ZX Spectrum 128, but it packs far more power

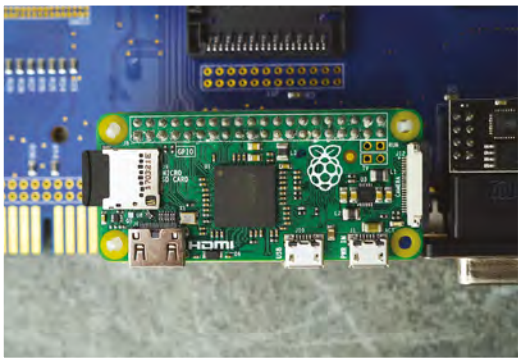
emulation: the programmable logic blocks create a perfect representation of the Z80 chip.

You can take the FPGA beyond the Z80 with processor cores. We turned our ZX Spectrum Next into a BBC Micro B and BBC Master using BeebFPGA (**magpi.cc/beebfpga**). Victor Trucco has made a range of Intel 8080 cores available, including MSX, NES, and Colecovision (**magpi.cc/specnextcores**).

A separate Anti-Brick core protect users from breaking the machine when messing around with cores, and can be used at any time to switch back to its original state.

> It's a great example of using the power of Raspberry Pi to add oomph to a project

Alongside this sits a Raspberry Pi Zero, which enables you to load digital .tzx files as analogue cassette tape (screeches, loading screen, and all). It also brings SID (Sound Interface Device) support to the table, enabling better audio for games. There are plans afoot for Raspberry Pi Zero's micro USB port to act as a digital joystick port, and the mini HDMI output may be used down the line to add a second display. Beyond that, Raspberry Pi Zero adds a 1GHz CPU and 512MB of RAM to the hardware – plenty of extra headroom for ambitious game developers.



▲ Attached to the motherboard, a Raspberry Pi Zero acts as a sound system, enabling files to be loaded as if they were tapes

We're impressed. From a design and build quality perspective, ZX Spectrum Next has achieved all we wanted from a new Spectrum. And it's a great example of using the power of Raspberry Pi to add oomph to a project. From a licensing and business perspective, managing to maintain this purity of focus while blending multiple open-source and proprietary software projects, all while juggling licensing owned by (to our count) 15 separate organisations including Amstrad/Sky, is seriously impressive stuff. Bravo, SpecNext, bravo!

Verdict

The ZX Spectrum Next is a lovely piece of kit. Well-designed and well-built: authentic to the original, and with technology that nods to the past while remaining functional and relevant in the modern age.

9/10

Picade

▶ Pimoroni ▶ **magpi.cc/iLOfHv** ▶ £150 / $159

The new Picade model is sleeker with a host of improved features. **Phil King** relives his misspent youth down the arcades

New versions of products are usually billed as 'bigger and better', but Pimoroni's new Picade is smaller than its original mini-bartop arcade cabinet. The display is still 8 inches (a 10-inch model is also available for £195 / $206), however. This time it's an IPS (in-plane switching) panel with wide viewing angles, higher resolution (1024×768 compared to the earlier 800×600), and a new Pimoroni-designed driver board with HDMI input and keypad controls for an on-screen menu.

Another key improvement is the new Picade X HAT, which works with any 40-pin Raspberry Pi. Also available separately (£15) for those who want to build their own custom arcade cabinet, the HAT has easy-to-use DuPont connectors for the numerous joystick and button wires. An additional 'Hacker' header breaks out the few remaining unused GPIO pins and I²C, which could be used to add extra buttons. The HAT also features power management and a built-in I²S DAC with 3W amplifier for mono audio – this time there's only one speaker included, although it's plenty loud enough.

◀ The new Picade is easier to build and looks fabulous sitting on your coffee table

▲ The Picade X HAT has easier-to-use connections, including a 'Hacker' header

Before you play on your new Picade, you'll need to assemble it. Taking two to three hours, this is an easier process than before, although there are still fiddly bits – mainly involving holding the tiny M3 nuts in hard-to-access places while screwing bolts

> Before you can play on your new Picade, you'll need to assemble it

(tip: use Blu Tack). Full instructions are supplied on the reverse of an A1 poster, but we found the appended online ones, with videos, easier to follow. Assembly is aided by some excellent packaging, with separate colour-coded boxes for the cabinet, screen, fixings, and accessories.

Arcade assembly

Firstly, a few of the black powder-coated MDF panels need to be screwed together with plastic brackets. Placed upside-down onto a clear acrylic panel, the screen display is connected to its rear-mounted driver board by a short flat flex ribbon cable and care needs to be taken not to pull out the connector tabs too far when inserting it.

Next, add the 30 mm push-fit arcade buttons and a microswitched joystick with ball top. Since these are standard parts, you could potentially customise your Picade by using different (possibly illuminated) buttons and joystick topper.

The wiring is easier than on the original Picade due to the DuPont connectors on the HAT, so you simply push in the pins of the wires, although the other ends still have spade connectors (and there are push-fit connectors for the speaker wires). As long as you get each wire loom the right way round at the HAT end, you should be able to make the correct connections for the joystick and buttons. In addition to the six control buttons, there are four utility buttons placed around the cabinet and a light-up yellow power button – simply press this to turn the Picade on and off, automatically shutting down Raspberry Pi safely – a really nice touch.

Playtime

Before turning on, you'll need to download RetroPie and write it to a microSD card – and uncomment a line in the **boot/config.txt** file to force HDMI output, to make the display work. The card can then be inserted into the Pi mounted inside the cabinet via a handy access hole. Alternatively, the back panel can easily be removed for easy access to all the components.

A 5 V USB-C power supply (not included) powers the Picade X HAT, which in turn powers Raspberry Pi, and the display via a USB cable. Hit the power button and away you go. Well, not quite. You'll first need to connect a keyboard to Raspberry Pi and install the Picade X HAT driver with a one-line Terminal command.

Then it's just a matter of setting up the joystick directions and buttons in the EmulationStation menu and – after adding files to RetroPie – playing your favourite homebrew games!

SPECS

SCREEN:
8-inch IPS panel, 1024×768 pixels

BOARD:
Picade X HAT

CONTROLS:
Joystick, 6 × arcade buttons

SPEAKER:
3-inch, 5 W, 4 Ω

DIMENSIONS:
350 × 230 × 210 mm

◀ Just like the one at your local arcade, only much smaller!

Verdict

A fun, if at times fiddly build, this all-new Picade features high-quality components and feels sturdy. Major improvements over the original version include a vivid, higher-res IPS display and easier-to-connect Picade X HAT.

9/10

RETRO COMPUTING

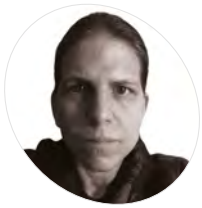
EMULATE CLASSIC COMPUTERS WITH RASPBERRY PI

> Back in the day, you could learn a lot by typing in the code given away with computer magazines


ESC INFO ENT
COIN SELECT
VALKYRIE
GAUNTLET

Play classic console games legally on Raspberry Pi

Discover a range of ways to buy and source classic games legally for Raspberry Pi



MAKER

K.G. Orphanides

K.G. is a writer, maker of odd games and software preservation enthusiast. They will fight anyone who claims that piracy is the only thing emulation's good for.

@KGOrphanides

Console emulation has been firmly in the mainstream in recent years. However, hobbyist emulation and DIY consoles run the risk of involving you with illegal copyrighted content.

But you don't have to be a bootlegger to build your own home multi-console emulation with Raspberry Pi and RetroPie.

Emulators themselves are strictly legal, and we've talked in the past about the wide range of homebrew and legal ROM images available (**magpi.cc/legalroms**).

In this tutorial we're going to look at a much broader range of legal console ROMs. Some can be purchased legally, while others have been developed and are distributed for free.

So let's set up a RetroPie console and play some classic games.

▲ Ecco the Dolphin is just one of the classic games in the Mega Drive Classics collections

Thriving scene

Sega's Mega Drive Classics collections include ROM images of the games that you run on any emulator you like, and brand new commercial releases for Sega and other platforms are thriving, as are active homebrew scenes bringing innovative new games to console formats that went out of production over 25 years ago.

Sega is incredibly supportive of its emulation community, and is happy to just sell you some classic Mega Drive ROMs, included in the Sega Mega Drive Classics collections for Windows, macOS, and Linux. You can buy 50 classic Mega Drive games on Steam (**magpi.cc/segaclassics**), either individually or as a pack.

Once bought, to find the ROMs, open the title's Steam Library page, clear the gear icon on the right, select properties Properties, select the Local files tab, and then click Browse local files. You'll find all the ROMs in the clearly labelled uncompressed ROMs directory. Rename all files with '.68K' and '.SGD' extensions to '.bin' and copy them over to Raspberry Pi using a USB stick or via its Samba share (**magpi.cc/samba**).

Buy new classics

If you're after new games for classic systems, **itch.io** should be your first port of call. It has legal homebrew games developed for popular 8-bit consoles. The Mega Drive has won the hearts of 16-bit devs, who develop new games for it (and other

16-bit consoles from that era). Be careful to get games that are sanctioned by the makers.

We've made itch.io collections for each of those platforms, going out of our way to avoid unauthorised ports and ROM hacks. These include both commercial and freeware games, plus a couple of open-source titles.

That's not the only place that you'll find releases for those platforms. In the tutorial, we download Mystery World Dizzy by the Oliver Twins. It's a wonderful example of a lost game that was recovered by its creators and released as freeware to the fan community, but it's also rare to find similar high-profile games from that era re-released with their creators' blessing. Unlike Sega, other large gaming companies don't look fondly on ROM hacks, fan games, and the like.

On the homebrew side of things, projects such as Retrobrews (**retrobrews.github.io**) and sites like **vintageisthenewold.com** and **indieretronews.com** compile collections and lists of homemade games for classic consoles, but watch out for the odd unauthorised port slipping into their catalogues.

There's a small but lively industry releasing cartridges for retro consoles, and a number of developers and publishers make the ROM files available online, either for free or a small price. Among these are RetroSouls (**retrosouls.net**), the team behind Old Towers, and Miniplanets publisher Playonretro (**playonretro.itch.io**).

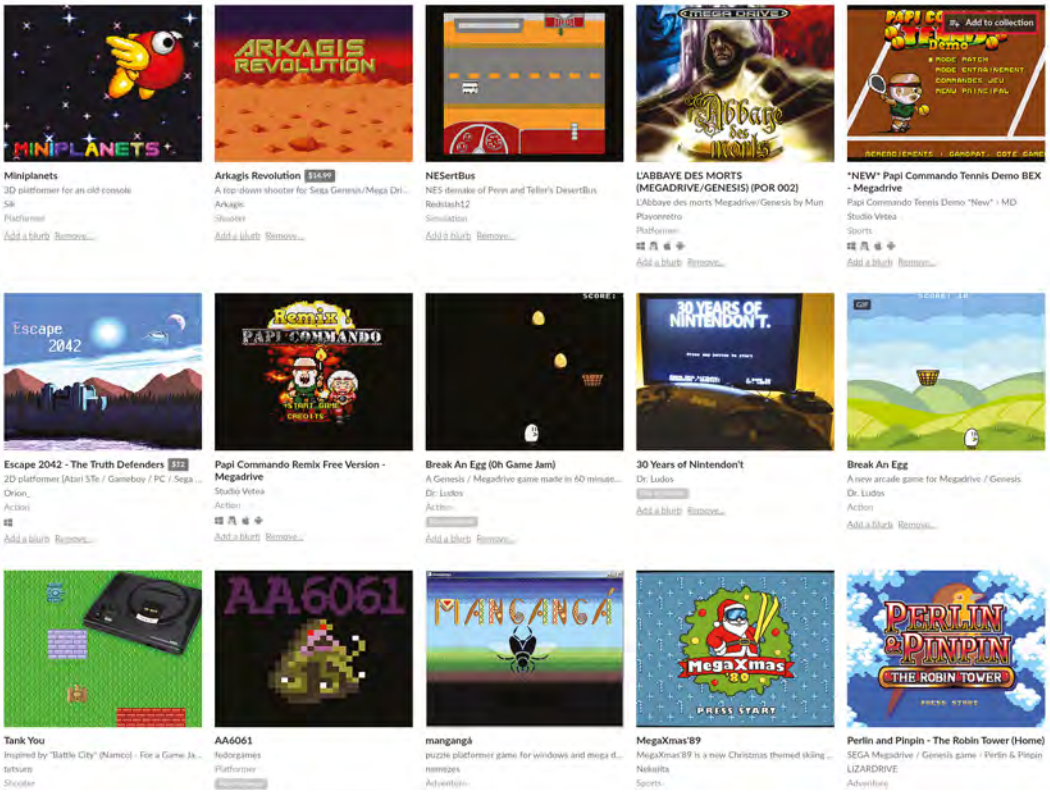
If you buy Sega's Mega Drive Classics collection on Steam, you'll get emulator-friendly ROM files for 50 games, including Golden Axe, Ecco the Dolphin, and Sonic the Hedgehog

> "Sega is incredibly supportive of its emulation community"

Eight modern Mega Drive games

Here are some of the best new Mega Drive games:

Tanglewood – magpi.cc/tanglewood
Miniplanets – magpi.cc/miniplanets
Devwill Too – magpi.cc/devwilltoo
Arkagis Revolution – magpi.cc/arkagis
L'Abbaye des Morts – magpi.cc/labbaye
Old Towers – magpi.cc/oldtowers
Irena: Genesis Metal Fury (demo) – magpi.cc/irena
Cave Story MD – magpi.cc/cavestory



New developers publish games for classic consoles on popular indie platform itch.io

Top Tip

Handheld paradise

If you'd rather build a handheld console, then that's a very viable prospect using a chassis such as the Retroflag GPi Case or the Waveshare Game HAT.

01 Image your RetroPie drive

Download the Raspberry Pi Imager for your operating system from **magpi.cc/imager**. Insert a microSD card – 8GB will be fine if you plan to limit yourself to 8- and 16-bit games, but if you want to emulate more powerful consoles in future, a 32GB card is a good investment.

Run Raspberry Pi Imager and pick RetroPie from the choose operating system list. You want the most powerful Raspberry Pi you can lay your hands on, but a Raspberry Pi Zero will do the trick if you stick to emulating relatively early systems, and is great for embedded console projects.

Choose your microSD card, click Write, and give the software permission to overwrite any data on the microSD card. Wait for the image to be downloaded and flashed.

Itch.io console games

Use these links to find new games for classic consoles on itch.io:

Mega Drive games: magpi.cc/itchmd
8-bit games: magpi.cc/itchnes
Master System games: magpi.cc/itchms
16-bit games: magpi.cc/itchsnes

02 Plug it in, baby

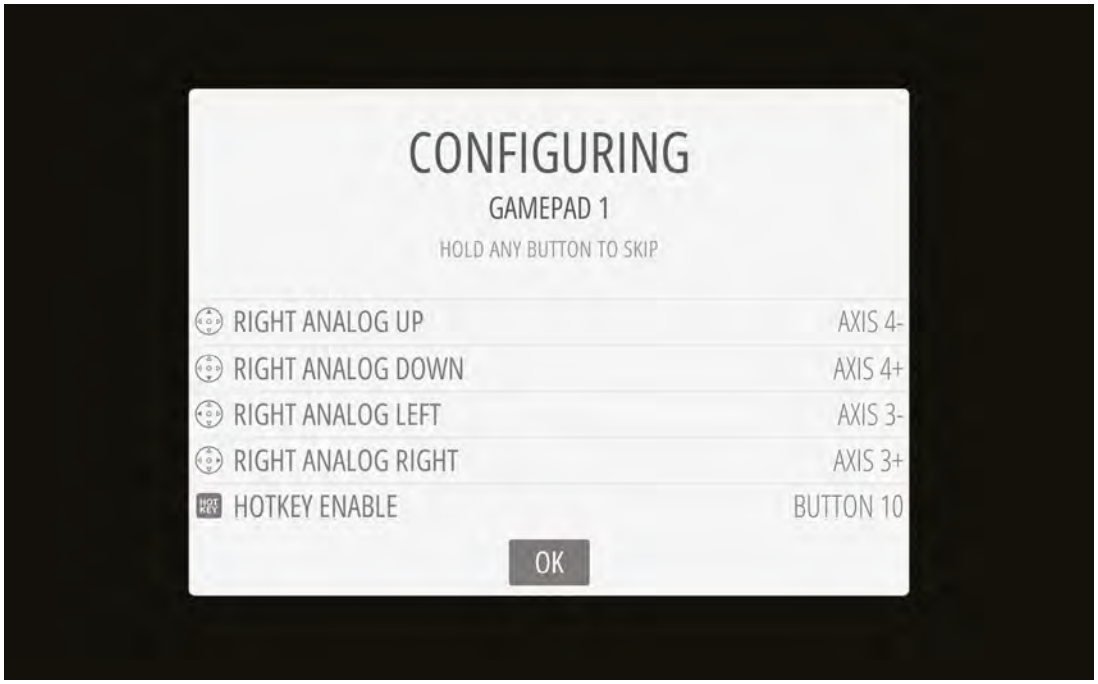
Insert the microSD card and connect Raspberry Pi to a keyboard, mouse, and monitor. If you've got controllers or joysticks, plug them in before you power up.

After the image boots, you'll be prompted to assign your gamepad's buttons, if you have one. Trigger buttons on some controllers – notably Xbox 360 compatible gamepads – may not register when pressed. Press and hold any other button to skip configuring them for now. If you make a mistake, you'll be able to go back and correct it when you get to the end of the configuration list.

03 Fix your triggers (optional)

If the triggers are unresponsive on your Xbox 360 compatible controller, you should update the xpad driver. Go to RetroPie configuration and select RetroPie Setup. From the ncurses menu, select Manage Packages > Manage Driver Packages > 847 Xpad Driver, then Update.

Exit back to the main EmulationStation interface and open the Menu. You may find that this has been remapped from Start to the Right Trigger button after the update. Scroll down and select Configure Input.



It takes a little getting used to, but EmulationStation's controller configuration tool means that RetroPie can handle almost any gamepad you want to use with it

04 Set me up

With your controller configured, you'll be taken to the main interface. You won't see any emulators on offer until you've copied over games for them to play. Press A on RetroPie to enter the config menu.

Select WiFi. Press OK at the following menu to go on to connect to a wireless network. Choose from the network list and enter the network key. Select Exit to return to the main EmulationStation config menu.

Some 1920×1080 displays will show a black border. If this is the case, select raspi-config. Go to Advanced Options, then Overscan – this will get rid of the black border. Select No to disable overscan compensation. You'll need to reboot for this to take effect.

05 Get some ROMs

Before we go any further, you'll need some games to run on RetroPie's suite of emulators. For our first ROM, we'll grab the Oliver Twins' Mystery World Dizzy. Go to **yolkfolk.com/mwd** and click Download. To test Mega Drive emulation, go to **arkagis.com** and click 'Download trial version' to take Arkagis Revolution's great rotating field navigation for a spin.

> It's easiest to download ROMs on another computer and copy them across

It's easiest to download ROMs on another computer and copy them across your local network to RetroPie's Samba share at **retropie.local** using your file manager. Each console has its own subdirectory under the **roms** directory. Windows users should ensure that network discovery is enabled.

Four modern 8-bit games

These new games are excellent examples of modern retro game development:

Micro Mages – magpi.cc/micromages
From Below – magpi.cc/frombelow
Wolfling – magpi.cc/wolfling
Legends of Owlia – magpi.cc/owlia

Micro Mages is a commercial modern game with fantastic graphics and tight single- and multiplayer gameplay for up to four people

Top Tip

What's a ROM?

ROM (read-only memory) files are data images of a non-rewritable storage medium, usually a game cartridge or – more rarely – computer or console firmware.

06 Time to play

Back on Raspberry Pi, restart EmulationStation: press Start on your controller, select Quit, then Restart System. Restart the interface every time you add games to force it to re-check its ROM directories.

If you have a keyboard connected, it's quicker to press and hold **F4** to quit to the command line, then type `exit` to restart EmulationStation.

As you scroll to the left or right, you should see a logo for the Mega Drive. Press A to enter the menu, then press A while highlighting the game you want to play. Right and left directional controls switch between different consoles.

Old Towers is a new homebrew Mega Drive game, available as a digital download or even as a cartridge!

07 Shortcuts, mods, and fixes

Remember the Hotkey you defined during controller configuration? You'll be using that a great deal, as it serves as a mode switch for controller shortcuts. You'll find more info at **magpi.cc/hotkeys**, but these are the most useful:

Hotkey + Start – quit the game
Hotkey + Right Shoulder – Save
Hotkey + Left Shoulder – Load
Hotkey + B – Reset
Hotkey + X – Open quick menu for save states, screenshots, recording and similar

If you don't get any audio from Raspberry Pi 4, make sure the HDMI lead connecting your monitor is plugged into the HDMI 0 port, nearest to the power connector.

Direct download ROMs

Although it's easiest to copy ROMs over from another computer, you can just download them at the command line of your RetroPie box if you have the URL. Press and hold **F4** to exit to the command terminal. You can download the ROM files directly to their directories using wget:

```
wget -P /home/pi/RetroPie/roms/nes/ http://yolkfolk.com/flash/mwdidd.nes
```

Restart EmulationStation by typing `exit` at the command prompt. If you'd rather just download all your files to a single location and move them later, the Midnight Commander file manager accessible from the Configuration menu makes this fairly hassle-free too.

Kits for Raspberry Pi 3, 4, Zero / Zero W, Zero 2 W and more!

RETRO GAMING MADE EASY

Get Gaming with everything you need all in one place at Vilros.com

Build a retro CD-ROM console

Connect a PC DVD-ROM drive and CRT TV to Raspberry Pi to play original disc games for 1990s computers and consoles

Working with original CD-ROMs is critical to software preservation, backup maintenance, and full emulation. This month, we'll add a disc drive to Raspberry Pi 4, connect a TV to make the most of CRT-era graphics, and overclock Raspberry Pi for an emulation performance boost

We'll use this hardware to add disc support to the system we made in our DOS emulation tutorial (see **magpi.cc/dosemulation**) and to emulate early disc-based consoles. We'll also explore the best legal landscape of disc emulation.

This project works best with Raspberry Pi 4 and a freshly installed Raspberry Pi OS (32-bit).

Images, discs, and the law

In the UK, you're not allowed to make copies of software, video, or music discs you've bought (**magpi.cc/ukgovcopying**); there are no exceptions (**magpi.cc/copyexceptions**) for backups or transcoding to play on another platform.

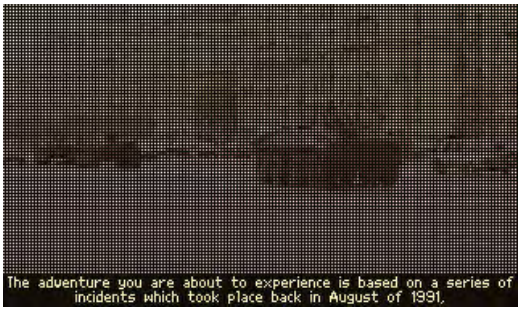
Unlike some PC software, permission to make copies for personal use is never granted in console games' End User License Agreements (EULAs). You have to use the original discs.

More obviously, you can't download disc images that someone else has made (even if you already own the game) or console operating system BIOS files. This means we'll be restricting ourselves to emulators that can actually play games from disc and which have a High Level Emulation (HLE) BIOS.

This peculiar combination of laws currently rules out a number of normally viable emulation platforms, such as the Amiga CD32, for which BIOS images are legally available via Cloanto's Amiga Forever (**amigaforever.com**), as the emulators that use them expect you to work with CD ISOs rather than original discs.

Similarly, although the RetroArch Disc Project (**magpi.cc/retroarchdisc**) is doing fine work on introducing disc support to certain Mega CD, Saturn and 3DO emulators, most of the emulators that currently have real disc support require BIOS images that you won't be able to legally obtain in the UK.

Read on, though, because that still leaves a few disc-based gaming platforms you can bring back to life with Raspberry Pi.



▲ Early DOS CD-ROM games like Conspiracy were designed to run directly from the disc

> This project is an excellent use for any old PC CD or DVD drives you might have

Disc support

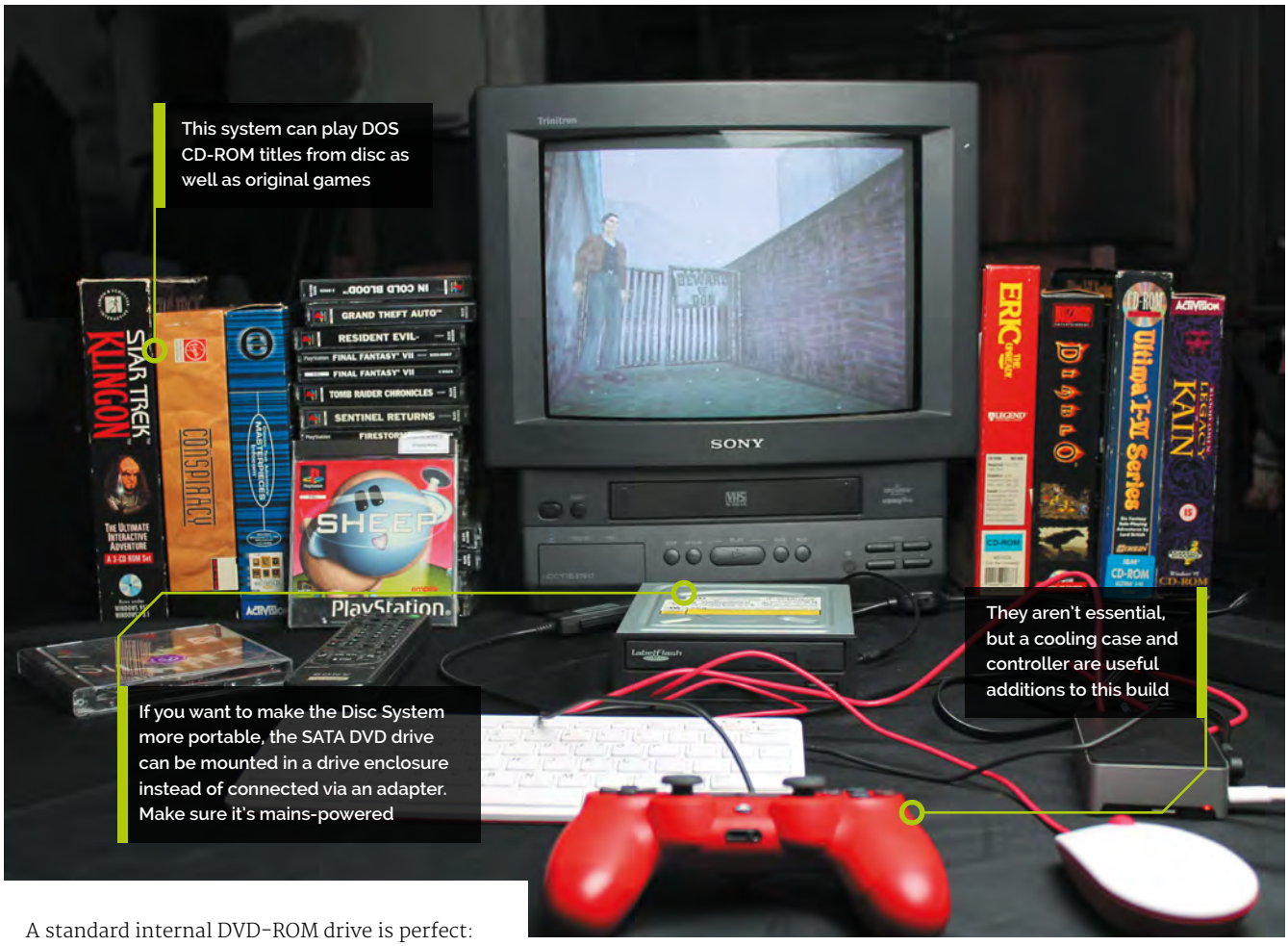
USB disc drives and Raspberry Pi can be an awkward combination. Modern bus-powered drives often use dual power/data USB connections that require more power than Raspberry Pi can readily supply, and don't play nicely with USB hubs or external 5 V power adapters.

Emulation adds to these problems, as early consoles often expected the disc to be spinning at all times, which many portable USB disc readers are unhappy with. Similarly, avoid Blu-ray drives: their spin and spin-down speeds frequently don't mesh well with the expectations of emulated consoles.

Warning! CRT

Cathode-ray tube television sets can be dangerous to repair. Be careful if opening up a CRT device.

magpi.cc/crt



A standard internal DVD-ROM drive is perfect: this build used a 2008 Sony NEC Optiarc AD-7203S SATA DVD-RW. Drives in this range are widely available for around £15, and this project is an excellent use for any old PC CD or DVD drives you might have lying around.

To connect it, you'll need either an external disc drive enclosure or SATA to USB adapter that takes external mains power. The kit photo above shows a StarTech USB2SATAIDE, which also supports IDE CD-ROM drives and hard disks. While this adapter is a little pricey at £42, similar hardware can be bought for about £20.

01 Connect your disc drive

Plug the SATA data and power connectors of your adapter into the back of your DVD-ROM drive, plug the adapter's USB connector into Raspberry Pi, and its mains adapter into a plug socket or power strip.

This also works with externally powered drive boxes, which look better if you want a tidy and portable final product, but will require a little more assembly to the tune of a few screws.

02 Overclock Raspberry Pi (Optional)

Emulation can be demanding, so GPU and CPU overclocking makes sense, although it's not absolutely necessary for this project. In a Terminal, type:

```
sudo mousepad /boot/config.txt
```

And add the following lines:

```
over_voltage=6
arm_freq=1750
gpu_freq=700
```

These were stable during testing, but if Raspberry Pi fails to boot, power-cycle it and hold down **SHIFT** to boot into recovery mode. Then knock the settings down a bit. See **magpi.cc/overclock** for further information on overclocking Raspberry Pi 4.

If you overclock, you should use a stand or, better still, an active or passive cooling case. A FLIRC Raspberry Pi 4 Case worked well here (**magpi.cc/flirc**).

You'll Need

- Full-sized internal desktop DVD-ROM drive
- Mains-powered SATA to USB2 adapter or drive enclosure
- Optional: Active or passive cooling Raspberry Pi case
- Optional: CRT TV

▲ The source code may be lost forever, but you can still play Silent Hill in its original glory – complete with tank controls

03 Enable OpenGL

We'll want OpenGL support for some emulators, such as PCSXR, In a Terminal, enter `sudo raspi-config`. Select Advanced Options > GL Driver > GL (Fake KMS), then exit and allow the system to reboot.

Open **/boot/config.txt** and make sure the following option is present and not commented out:

```
dtoverlay=vc4-fkms-v3d
```

04 Drop your resolution

Dropping your display resolution is an easy way of improving emulator performance. If you're using a standard 1920×1080 widescreen monitor, you won't need that resolution to play older games.

Open the menu and go to Preferences > Screen configuration and set your resolution to 720×576 (or 640×480) if you either have a 4:3 display or can live with a bit of screen stretching in exchange for smooth full-screen graphics. Choose 1280×720 if you don't mind playing in a window on emulators that can't do aspect ratio correction.

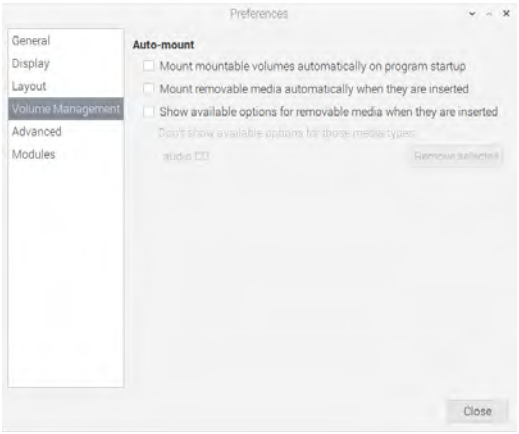
05 Connect an elderly TV (Optional)

A 4:3 aspect ratio display is ideal here, and older display tech has the edge for 1990s console and computer games, too.

Using a CRT TV rather than a modern LCD flatscreen can improve graphical quality as sprite and even 3D graphics of the era were optimised to work with the display artefacts of CRT.

Raspberry Pi supports composite video out. Connect a 4-pole 3.5mm AV cable to the 3.5mm port on Raspberry Pi and connect the other end to your TV. Using a composite to SCART adapter can improve picture stability.

Note that Raspberry Pi's 4-pole connector expects video to be connected to the sleeve and ground to ring 2, so ensure that you use a fully compatible cable (**magpi.cc/monitorconnection**). The wrong cable selection can result in non-functional sound, misordered cables, or even damage to your hardware.



▲ To control fixed-path disc mounting for DOSBox using pmount, you'll have to disable the File Manager's default volume management behaviour

> "Dropping your display resolution is an easy way of improving emulator performance"

06 Output composite video (Optional)

If you're using a typical 4:3 PAL TV, make the following changes to **/boot/config.txt** to correctly position your display – small alterations may be required for different models.

```
disable_overscan=0
overscan_left=16
overscan_right=16
sdtv_mode=2
```

Quick Tip

Clean your discs

The condition of discs can lead to choppy sound and video identical to a disc drive that's spinning too fast for the emulator, so use a clean, unscratched game while testing.

In a Terminal, enter `sudo raspi-config`, then go to Advanced Options, Pi 4 Video Output, and Enable analogue TV output. Finish and reboot.

07 Mount a CD in DOSBox

If you've been following these tutorials, you may already have DOSBox or DOSBox-X installed. If not, at a Terminal:

```
sudo apt install dosbox
dosbox
```

To mount a disc at the DOS prompt, type:

```
mount D /media/YourDiscName/ -t cdrom -usecd
0 -ioctl
```

To unmount a disc in DOSBox, type:

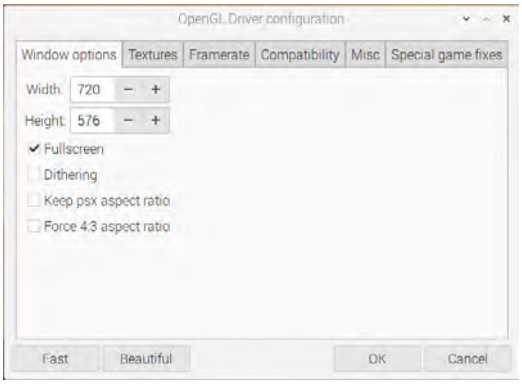
```
mount -u D
```

By default, each individual disc has to be manually mounted in DOSBox, as mount point names are automatically generated based on the volume name of the disc. This can be a problem if you need to swap DOS CDs during play or installation.

08 Create a fixed mount point

To work around this, we can use the `pmount` command. From the Terminal, let's first make sure it's installed and then configure it:

```
sudo apt install pmount
sudo mousepad /etc/pmount.allow
```



If you're going to be playing on a PAL CRT television, you'll need a 720×576 full-screen resolution

Add the following line to the file, then save and exit:

```
/dev/sr0
```

On the desktop, open File Manager. Go to Edit > Preferences > Volume Management and untick all the Auto-mount options. Reboot Raspberry Pi.

09 Mount and swap CDs

Now, to mount a disc, insert it, open a Terminal window and type:

```
pmount /dev/sr0
```

To unmount it:

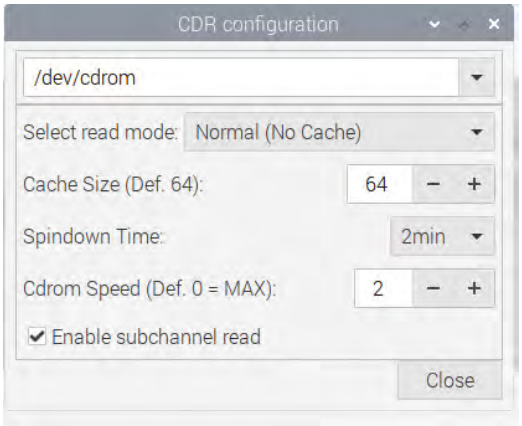
```
pumount /dev/sr0
```

Repeat the first `pmount` command to mount a new disc. Now, every disc will have a fixed mount point of **/media/sro/**. This means that, in DOSBox, you'll just need to mount **D /media/sro/** once.

When you want to swap discs, whether at the DOS prompt or in-application, hop over to a Linux Terminal window, run through the `pmount` commands and then, back in DOSBox, press **CTRL+F4** to update cached information about your mounted drives.

10 Play original discs

PCSXR – the R stands for either Reloaded or ReARMed, depending on which version you're using – is an open-source emulator.



Getting PCSXR's disc drive settings right is critical. Too high a speed or too fast a spindown can make games judder and stutter

Quick Tip

Composite Zero

Raspberry Pi Zero lacks its siblings' composite video out port, but instead has a TV header which you can use to connect an RCA cable. For detailed instructions, see **magpi.cc/rcapizero**.

▲ This build included an internal PC DVD-ROM drive, an externally powered SATA-to-USB adapter, a composite video out cable and SCART adapter, a heat-sink case, and controller

It also has a genuinely good emulated bios, so you don't need to download anything dodgy to make it work. The desktop version works best for original discs. Open a Terminal and type:

```
sudo apt install pcsxr
```

It can run games including Final Fantasy VII, Silent Hill, GTA, Sheep, and Resident Evil either perfectly or with only minor errors, but you'll have to adjust some settings first.

11 Configure PCSXR's graphics

Go to the Configuration menu and select Plugins & BIOS. From the Graphics pull-down, select OpenGL Driver 1.1.78. Click on the window icon directly to the right of the pull-down.

Starting with the Windows options tab, assuming you're using the PAL resolution we configured, enter a width of 720, a height of 576, and tick the Fullscreen box. On the Textures tab, set Quality to Don't care, Filtering to None, and HiRes Tex to None.

In the Framerate tab, ensure that 'Use FPS limit' is ticked and set to auto-detect. Moving to the Compatibility tab, select Standard offscreen drawing, a Black framebuffer, and Emulated Vram for framebuffer access. Make sure the Mask bit and Alpha multipass boxes are ticked.

In the Misc tab, tick Untimed MDECs, Force 15 bit framebuffer updates, and Use OpenGL extensions. The 'Special game features' tab includes game-specific options, such as battle cursors for Final Fantasy VII. Click Okay to save your changes.

12 Configure PCSXR's sound and CD-ROM

Click on the window icon next to the Sound pulldown in the configuration window. Set Volume to Low, Reverb to Off, and Interpolation to None. Unsick everything except Single channel sound. Click Close, then open the CD-ROM settings.

Set read mode to Normal (No Cache), Spindown time to 2 minutes, Cdrom Speed to 2min, and tick Emulated subchannel read.

While you may need to adjust these settings for individual games or experiment with higher resolutions, this combination allows the vast majority of titles to run reasonably smoothly from their original discs.

To test this, insert a disc into the drive, wait for it to load, then click on the CD icon at top left of the PCSXR window.

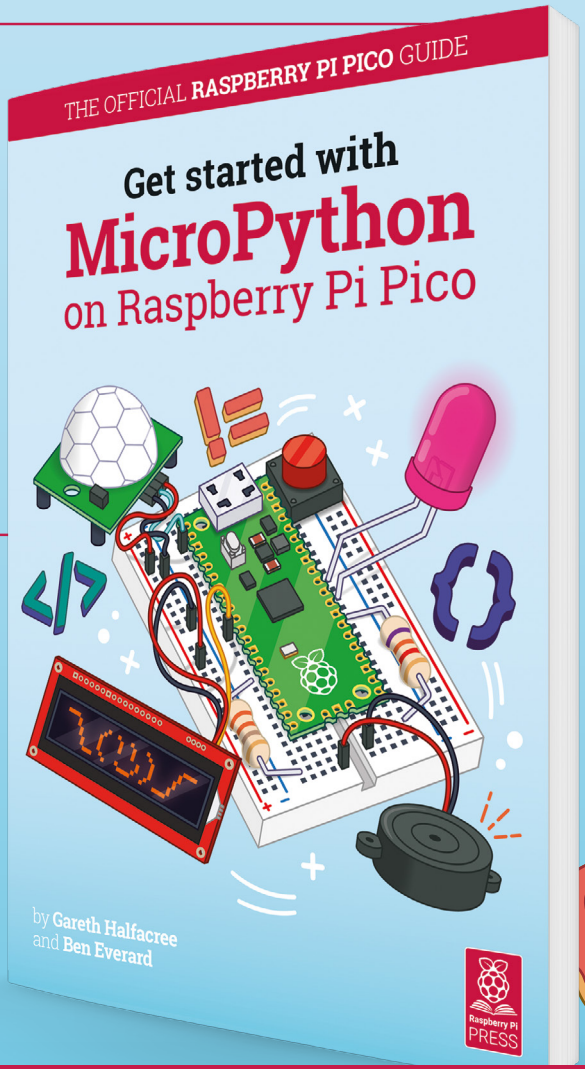
Quick Tip

Stand it up

If you don't have a cooling case, you can increase airflow around Raspberry Pi by using a stand to support it on edge.

Get started with MicroPython on Raspberry Pi Pico

Learn how to use your new Raspberry Pi Pico microcontroller board and program it using MicroPython. Connect hardware to make your Pico interact with the world around it. Create your own electro-mechanical projects, whether for fun or to make your life easier.



- **Set up your Raspberry Pi Pico and start using it**
- **Start writing programs using MicroPython**
- **Control and sense electronic components**
- **Discover how to use Pico's unique Programmable IO**

Available now: magpi.cc/picobook

Build a handheld console

Forget the TV, a dedicated handheld is the pinnacle of retro gaming. Whether you play in bed or in the garden, we've got builds for under £100 and under £200

This month, we're going to build handheld consoles powered by two different Raspberry Pi computers, in two different cases. The Retroflag GPi Case for Raspberry Pi Zero, supplied by The Pi Hut, has a total of eight buttons plus a digital pad and a 2.8-inch 320×240 colour screen. It'll run for hours on three AA batteries and is small enough to carry in a generously sized coat pocket.

The PiBoy DMG is more expensive, chunkier, and much more powerful. It has a 3.5-inch 640×480 display, both digital and analogue controls, and a total of ten buttons. You can access all Raspberry Pi 4's USB ports and there's an optional mini-HDMI pass-through. It won't run off AAs, so the full kit ships with a 4500 mAh rechargeable battery.

Alert! Copyright

Many classic games are protected by copyright. Stick to homebrew and legal ROMs.

magpi.cc/legalroms

You'll Need

- Retroflag GPi case (£60) **magpi.cc/gpi**
- microSD card (8GB+)
- Raspberry Pi Imager **magpi.cc/downloads**
- Monitor, USB-to-micro USB adapter, keyboard (briefly)
- 3 × AA batteries

Build 1: Raspberry Pi Zero

01 Install RetroPie

Use the Raspberry Pi Imager for Windows, Linux, and macOS to download and write RetroPie (RPI 1/ZERO) on a microSD card. 8GB capacity should be fine for our purposes, as none of the systems we'll be emulating involve large files.

Before you install Raspberry Pi Zero in the GPi Case, you connect it to a monitor, a keyboard, and the internet to install Retroflag's safe shutdown script. Insert your microSD card and connect your peripherals. Allow RetroPie to boot, then press **F4** to quit to the command prompt.

02 Basic config & safe shutdown

Type `sudo raspi-config`. Now go to Network Options. Go to Wi-Fi and set your country, network name (SSID), and password. In Localization options, Change Keyboard to make sure your keyboard is properly configured. **TAB** to Finish on the main menu, press **ENTER**, and reboot.

At the command prompt, type `ifconfig` to confirm that your wireless network is connected Finally, on a single line, type:

```
wget -O - "https://raw.githubusercontent.com/RetroFlag/retroflag-picase/master/install_gpi.sh" | sudo bash
```

This will download and run the safe shutdown installer before restarting Raspberry Pi Zero. Power down and unplug the system.

03 Add display support

Return the microSD card to the system you're using to prepare the OS for use. Download the patch zip file from **magpi.cc/gpicasepatch** and unzip it.

The readme file includes instructions for Windows and macOS – the latter also applies to Linux operating systems including Raspberry Pi OS.

From the boot partition of your RetroPie disk, copy **config.txt** to the **original_files** directory in the patch's folder and replace it with the one that you'll find in the **patch_files** subdirectory.

Similarly, back up **dpi24.dtbo** from RetroPie's **/boot/overlays** folder to the supplied **overlays** directory, then copy over **dpi24.dtbo** and **pwm-audio-pi-zero.dtbo** from the **patch_files** subdirectory to RetroPie's **overlays** folder.

04 Prepare the case

The Retroflag GPi Case comes with a helpful illustrated installation guide, a USB power cable, plus the screwdriver and four screws you'll need to assemble your handheld.

The Retroflag GPi will just about fit in a jeans pocket, so you can play From Below wherever you go

The larger screen size and chunkier dimensions of the PiBoy DMG make it feel like playing a horizontal console despite its vertical form factor

Open the battery compartment at the back and flip the Safe Shutdown switch to the 'on' position. Make sure the main console power switch is in the off position.

Remove the 'cartridge' – actually a Raspberry Pi Zero case – from the slot at the top of the console; turn it so that the sticker's facing you and gently but firmly pull it apart.

Remove the microSD card from Raspberry Pi and the microSD cover from the case.

05 Install Raspberry Pi

Place Raspberry Pi loosely into position on the four mounting posts in the shell, with the SD slot facing the gap you removed the cover from.

Connect the micro USB extension ribbon cable from the I/O conversion board that comes installed in shell 2 to Raspberry Pi's USB port (the rightmost – the other one is only for power). Now seat Raspberry Pi into shell 1 and position the I/O board on top of it. Make sure both the posts and GPIO pogo pins are lined up.

Reinsert the microSD cover, clip the cartridge halves back together, and install the supplied screws into the holes on the back to secure it. Open the SD card cover, insert your card, close it, and slide the cartridge back into the main body of the case. Insert three AA batteries.

A micro USB port under the GPi's battery cover can provide USB peripheral connectivity via a powered hub, but reliability varies greatly from hub to hub

06 Power up and configure

Flip the power switch at the top right and RetroPie will boot. The GPi Case registers as an Xbox 360 pad, less a few buttons, though left and right buttons are hidden on the back of the case.

Hold any button to start configuration. When you get to a button that doesn't exist, press and hold any button. Skip hotkey configuration and allow RetroPie to auto-configure it as Select when prompted. You'll be able to exit to the menu from games by pressing Start and Select at the same time.

In the front end, tap A to enter the RetroPie menu, scroll to RetroPie Setup, and tap A. Go to Configuration / tools, select Samba, and Install RetroPie Samba share to create a network share so you can easily copy game files over to the console's **~/RetroPie/roms** directory. RetroPie Setup also allows you to install new emulators.

Top Tip

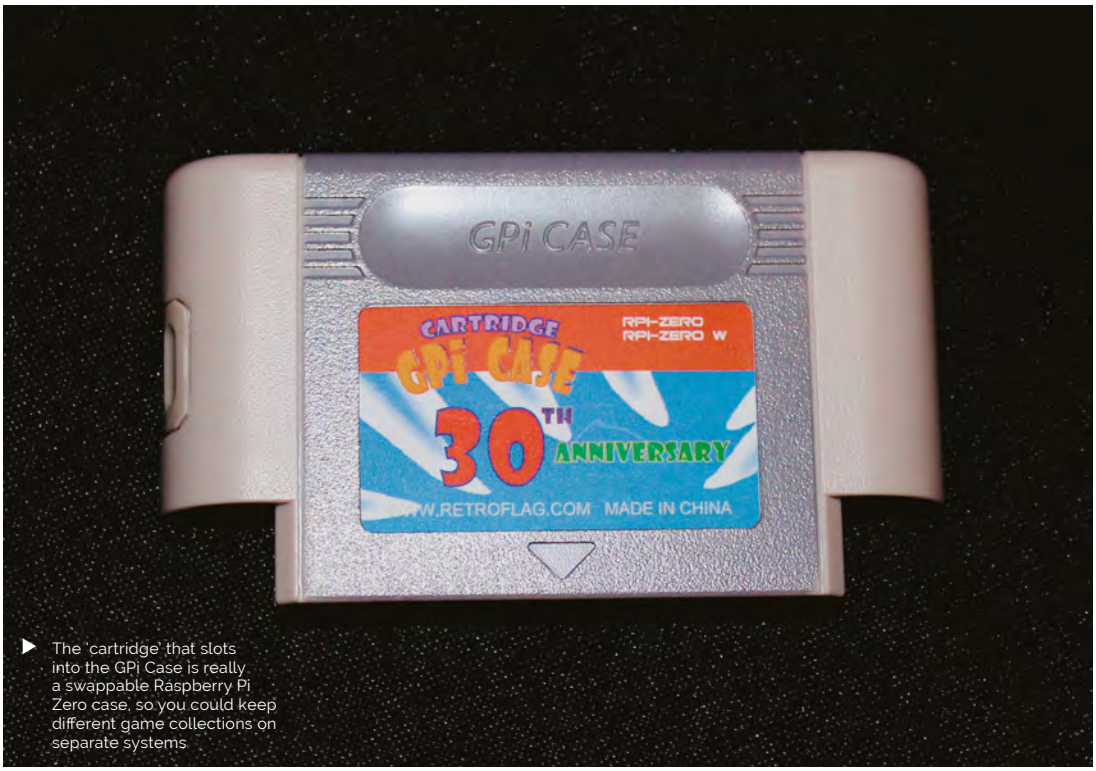
The right emulator

For improved Raspberry Pi Zero emulation, use lr-picodrive for Mega Drive, lr-pce-fast for PC Engine, and lr-genesis-plus-gx for Master System.

Top Tip

Windows required

PiBoy DMG kits should have the latest firmware, but future updates (**magpi.cc/piboyfirmware**) will require a Windows PC.



The 'cartridge' that slots into the GPi Case is really a swappable Raspberry Pi Zero case, so you could keep different game collections on separate systems

You'll Need

- PiBoy DMG – Full Kit ($120) **magpi.cc/piboydmg**
- Optional PiBoy DMG HDMI adapter ($10)
- microSD card (32GB+)
- Raspberry Pi Imager **magpi.cc/downloads**

Build 2: Raspberry Pi 4

01 Image your microSD card

Experimental Pi has its own fork of RetroPie, tweaked to fully support the handheld's features. Download and extract the operating system image via **magpi.cc/piboydmgimage** and flash it to your microSD card using the Raspberry Pi Imager tool.

Alternatively, you can install RetroPie – or any other Raspberry Pi OS / Raspbian-based operating system – but will have to add Experimental Pi's safe shutdown and on-screen display scripts, available at **magpi.cc/piboydmgscripts**.

Even on a backlit colour screen, modern 8-bit games feel right on a handheld

02 Chassis preparation

Experimental Pi's illustrated online assembly instructions for the PiBoy DMG are among the best we've seen, so keep them on hand during this build: **magpi.cc/piboydmgbuild**. The PiBoy DMG Full Kit comes with the battery, screws, screwdriver, buttons, and blanking plates that you'll need to build it. It's worth adding the HDMI adapter to your order, too.

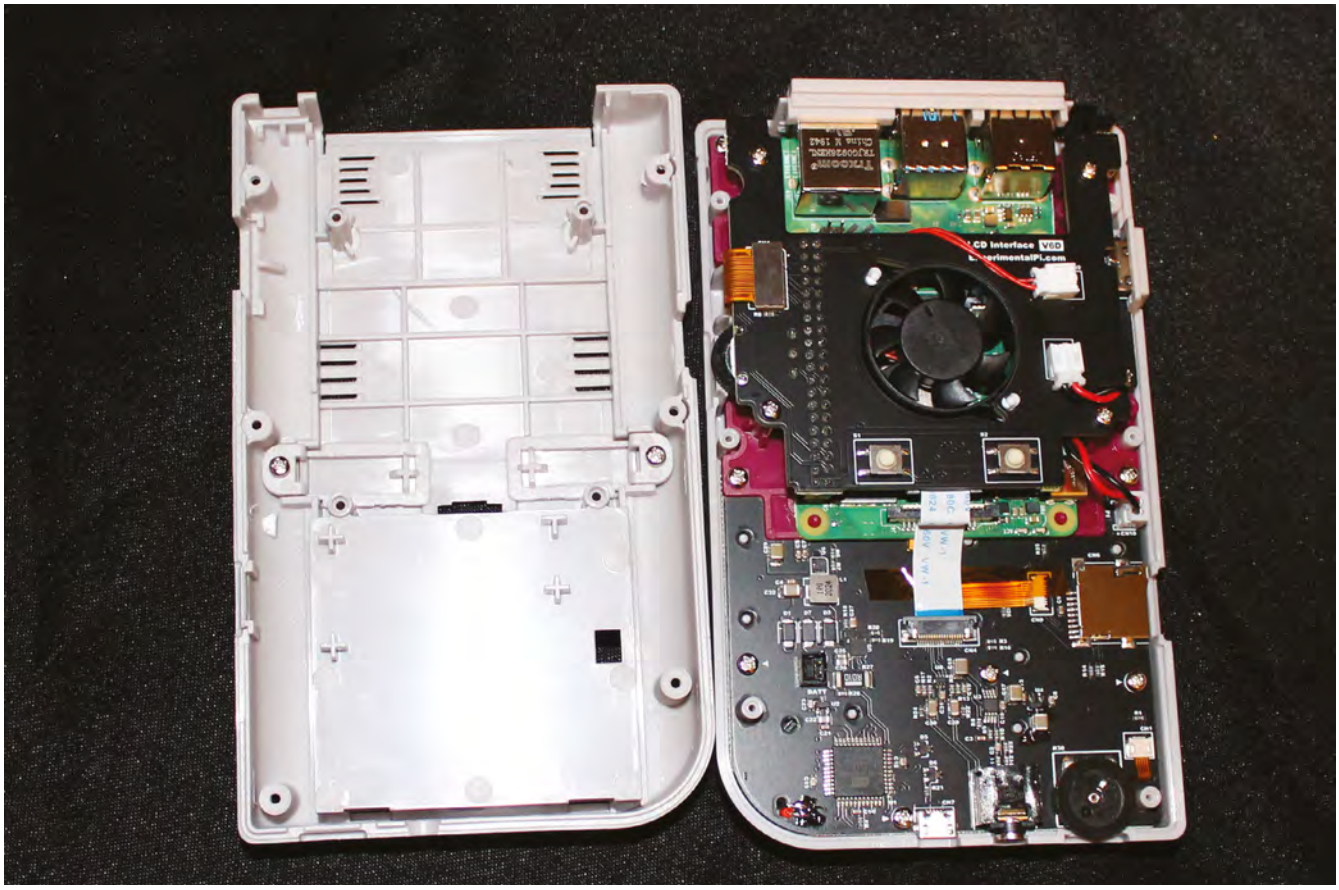
Unscrew the back of the case, and then unscrew and lift off the fan assembly that'll keep Raspberry Pi cool in situ.

> "It's worth enabling Samba for ease of transferring software to the console"

03 Install Raspberry Pi

Slip the supplied faceplate over Raspberry Pi's ports – and, if you're using it, fit the PiBoy HDMI adapter to the rightmost micro-HDMI port and slide its faceplate on.

Gently push the SD card adapter ribbon cable into Raspberry Pi's microSD slot, then lower the computer and HDMI adapter onto the standoffs. Screw the HDMI adapter into position. If you're not going to use this adapter, fit a blanking plate in its place.



▲ The Pi Boy DMG case has a built-in fan, which makes it a little noisy but also means you can overclock it to run demanding games

04 Fit the fan

Add the fan board: making sure that it's lined up with the GPIO, gently seat it into place – a rocking motion works well for this. Make sure all cables are correctly lined up and screw the board down. Line up and gently press into the place the IPS screen cable.

Place the supplied power switch onto the switch on the top right of the board and screw the rear of the case back on. Fit the rechargeable battery – it'll only connect one way round, but there are also polarity markings to help.

05 Go wireless

Slide the microSD card you imaged earlier into position and power up. To add wireless networking to our handheld build, mount its microSD hard disk on any other computer. In the top-level **/boot** directory, create a file called **wifikeyfile.txt**. It should contain these lines:

```
ssid="wifi_name"
psk="password"
```

Save the file, unmount the card, return it to your handheld, and boot. From the RetroPie menu, select 'Wifi', then import WiFi credentials from **/boot/wifikeyfile.txt**.

As with the Retroflag build, it's worth enabling Samba for ease of transferring software to the console (see Build 1, Step 6). Transfer your games, and you're ready to play on the move.

Handheld homebrew

To help you find the latest games for your favourite classic handhelds, here are six more game collections on indie platform itch.io.

- **magpi.cc/itchgb**
- **magpi.cc/itchgbc**
- **magpi.cc/itchgba**
- **magpi.cc/itchlynx**
- **magpi.cc/itchpsp**
- **magpi.cc/itchpce**

Top Tip

Why HDMI?

Adding the micro-HDMI adapter to the PiBoy DMG means you can connect it to a TV, add a couple of USB controllers, and enjoy classic multiplayer gaming.

Use a retro DB9 joystick with Raspberry Pi 400

Get the classic ZX Spectrum experience with Raspberry Pi 400 and a GPIO joystick

Raspberry Pi 400 is, with its integrated keyboard and tweaked performance, a modern home micro. This month, we pay tribute to one of the computers that helped inspire its look by emulating Sinclair's ZX Spectrum, complete with the latest games and retro joysticks connected via GPIO. These use the D-sub connector popularly called DB9, also known as DE-9.

Warning!
Copyright

Many (but not all) Spectrum games can be legally downloaded. See our Legal ROMS page for more information. **magpi.cc/legalroms**

You'll Need

- DB9 joystick
- DB9 breakout board **magpi.cc/db9breakout**
- 10–20 male-to-female jumper cables **magpi.cc/mfjumpers**
- DB9 GPIO driver source **magpi.cc/db9gpiogit**

01 Install the Free Unix Spectrum Emulator

Also available in the RetroPie emulation distro we've used in previous tutorials, the FUSE ZX Spectrum emulator can be found in Raspberry Pi OS's standard repositories, so installation is a bare minimum of fuss. Open a Terminal window and type:

```
sudo apt install fuse-emulator* spectrum-roms opense-basic libspectrum8
```

This will install FUSE, its GTK and SDL front ends, and both open-source system ROMs and the original Spectrum system ROMs. Permission has been granted for free modification and distribution of the latter (**magpi.cc/SpecROMs**).

While the open-source ROM can only handle a limited selection of files, that spectrum-roms package will allow you to play a wide array of games, including the latest generation of technically spectacular releases for the platform.

02 Test FUSE

Next, we'll make sure FUSE is working with the ZX Spectrum port of Locomalito's open-source classic, L'Abbaye Des Morts. In a Terminal, type:

```
wget https://spectrumcomputing.co.uk/zxdb/sinclair/entries/0030109/AbbayeDesMorts.tzx.zip
```

```
fuse-sdl
```

Press **F2** to open FUSE's file browser, navigate to **AbbayeDesMorts.tzx.zip**, and press **ENTER**. The game should load and run automatically. For full-screen mode, press **F1**, go to Options, and select 'Full screen'. Take note of the Filter option in the same menu, which lets you choose the emulator's upscaling method: 'TV 3x' gives you some pleasing scan lines and the correct aspect ratio.

> "Wiring up the DP9 port to Raspberry Pi's GPIO is a fairly simple process"

03 Wire up your joystick port

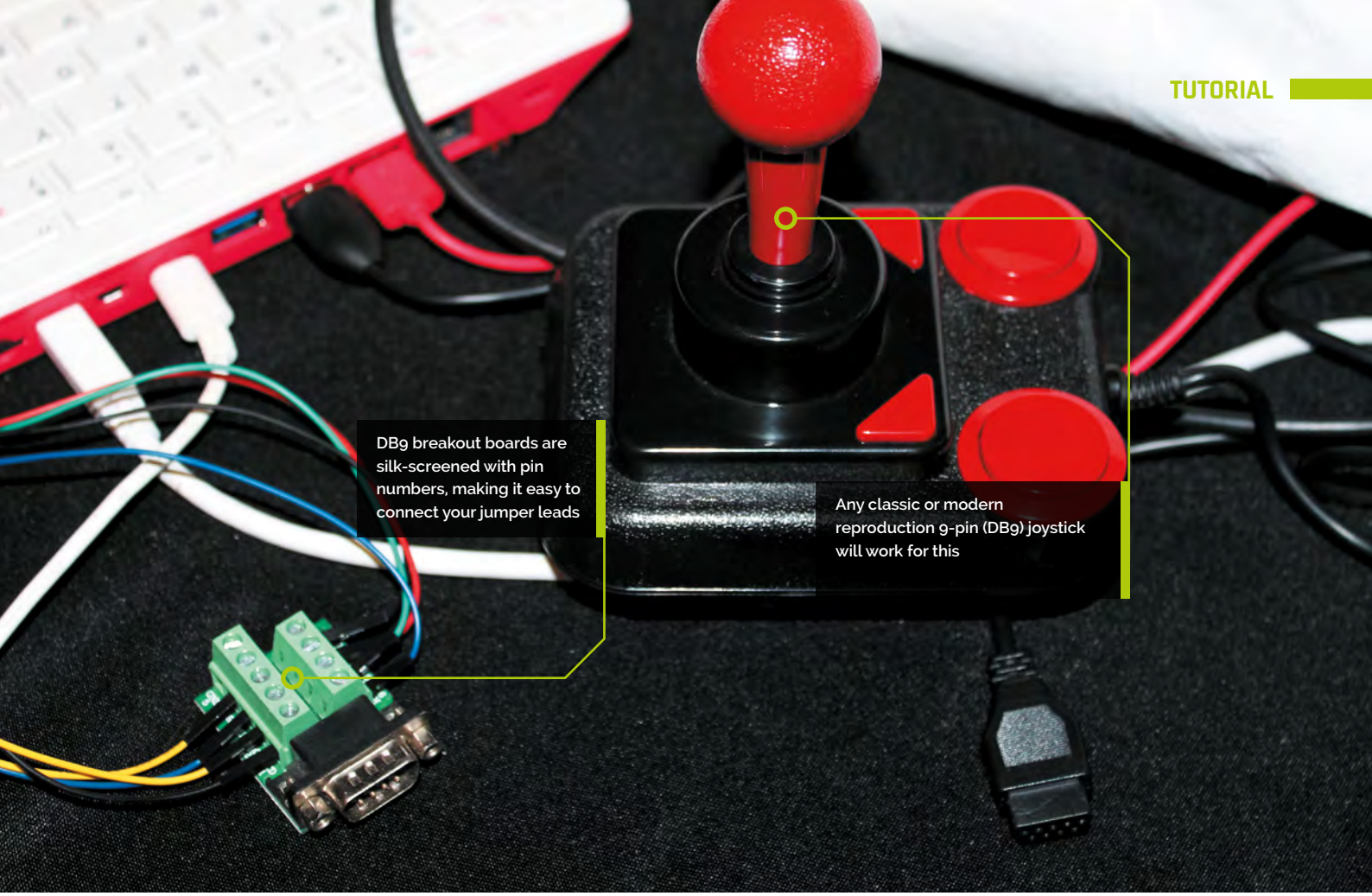
Standard DB9 connectors split the nine pins of your cable into nine screw-down terminals. We found it most convenient to use male-to-female jumper cables for this, clamping the male tips into our DB9 breakout connector.

For a classic single-button joystick like the Competition Pro Retro we used, pin 1 is up, pin 2 is down, pin 3 is left, pin 4 is right, and pin 6 is fire. Pin 8 connects to ground – we recommend using a green cable for it. Some joysticks may require you to connect port 7 to a 3.3V power connector on the GPIO, but the Competition Pro does not.

See the 'DB9 pins' box (overleaf) for an at-a-glance DB9 connection table.

04 Wire up Raspberry Pi

Wiring up the DB9 port to Raspberry Pi's GPIO is a fairly simple process, although you'll have to do some careful pin counting. On Raspberry Pi 400, pin 1 is at the top right of the horizontally



oriented GPIO port and pin 40 is at the bottom left. Remember that GPIO numbers don't correspond with pin position numbers.

For a reminder of where everything is, open a Terminal and type: `pinout`.

Our diagram (**Figure 1**, overleaf) shows where the female jumpers connected to your DB9 port need to go on Raspberry Pi. For a single one-button joystick, up goes to GPIO 4, down to GPIO 7, left to GPIO 8, right to GPIO 9, and fire to GPIO 10.

05 Build the DB9 joystick driver

Let's build the driver. Enter this in a Terminal window:

```
sudo apt install dkms raspberrypi-kernel-headers
git clone https://github.com/marqs85/db9_gpio_rpi.git
cd db9_gpio_rpi
sudo cp -r db9_gpio_rpi-1.2 /usr/src/db9_gpio_rpi-1.2/
sudo dkms add db9_gpio_rpi/1.2
sudo dkms build db9_gpio_rpi/1.2
sudo dkms mkdeb db9_gpio_rpi/1.2 --source-only
sudo modprobe --first-time db9_gpio_rpi map=1,1
```

pullup.sh

DOWNLOAD THE FULL CODE: magpi.cc/pullupfix

Language: **Bash**

```
001. #!/bin/bash
002. # fix for db9_gpio_rpi driver issue https://github.com/marqs85/db9_gpio_rpi/issues/8
003. # RPi 4 and 400 need this tweak to speak to db9_gpio_rpi gpio connected controllers as some inputs need explicit pullup
004. # ensure that you're applying the pullups to the correct pins - this is for a standard deployment of the driver
005. #
006. # Use:
007. # chmod pullup.sh +x
008. # add /path/to/pullup.sh to /etc/rc.local to load on boot
009.
010. # Joyport /dev/input/js0
011. raspi-gpio set 4 ip pu
012. raspi-gpio set 7 ip pu
013. raspi-gpio set 8 ip pu
014. raspi-gpio set 9 ip pu
015. raspi-gpio set 10 ip pu
016. raspi-gpio set 11 ip pu
017. raspi-gpio set 14 ip pu
018. # Joyport /dev/input/js1 - if we connect a second joystick
019. raspi-gpio set 15 ip pu
020. raspi-gpio set 17 ip pu
021. raspi-gpio set 18 ip pu
022. raspi-gpio set 22 ip pu
023. raspi-gpio set 23 ip pu
024. raspi-gpio set 24 ip pu
025. raspi-gpio set 25 ip pu
```

DB9 pins

For a one-button joystick – including multi-button sticks with only single-button functionality – connect these pins on your DB9 breakout board.

DB9 pin	Joystick function
1	UP
2	DOWN
3	LEFT
4	RIGHT
5	-
6	FIRE
7	-
8	GROUND
9	-

That `map` option defines what kind of joystick you're using, with each number classifying a different type of joystick As we're using a one-button joystick, `map=1,0` would do it, but `1,1` means we can connect a second stick of the same type to a second port.

> Although many Spectrum games support joysticks, you'll often have to enable support for these

Top Tip

GPIO reference

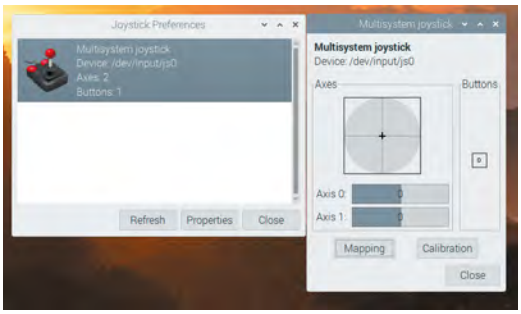
If you're flummoxed by GPIO pin numbering, see the official docs at **magpi.cc/gpio**.

06 Test your joystick

Building and loading the driver won't quite get us to a functional joystick, as the driver isn't fully compatible with Raspberry Pi OS's recent use of raspi-gpio instead of gpio. However, this is a great time to test your joystick to make sure that it's wired up correctly.

```
sudo apt install jstest-gtk
jstest-gtk
```

You should see your joystick in the peripherals list. When you click into it, if you're using a Competition Pro Retro or similar joystick, you're likely to find that the fire button is jammed on and that the horizontal x axis is stuck at the left.



▲ The jstest-gui program allows you to test your joystick and also indicates whether it's working as it's supposed to

07 Pull-ups are good for you

This is because your GPIO needs to be set up to handle the joystick's pull-up switches. On a standard DB9 GPIO configuration, the script you'll find at **magpi.cc/pullupfix** will do this for you. Create it using your preferred text editor and save it somewhere handy. We've put ours in **/home/pi/pullup.sh.** Test it by running:

```
sh /home/pi/pullup.sh
jstest-gtk
```

If the joystick is now aligned properly and the button isn't stuck on, you're in business.

```
chmod +x /home/pi/pullup.sh
```

Finally, let's load those pull-up settings on boot.

```
sudo mousepad /etc/rc.local
```

Above the `exit` line, enter the following:

```
/home/pi/pullup.sh
```

You can also place the pull-up code directly in **rc.local** if you wish.

08 Load on boot

Once you're satisfied that your joysticks work, you'll want to load the driver module on boot.

```
sudo mousepad /etc/modules
```

…and add:

```
db9_gpio_rpi
```

After saving and exiting the file, enter:

```
sudo mousepad /etc/modprobe.d/db9.conf
```

This file should contain the following line:

```
options db9_gpio_rpi map=1,1
```

If you're using a different joystick and configuration, you'll need an appropriate map, and possibly some extra GPIO connections, which you can find at **magpi.cc/db9gpio**.

Reboot Raspberry Pi and use `jstest-gtk` to ensure that everything is working as it should. You can now use the driver as a generic controller input device.

09 FUSED joysticks

FUSE doesn't enable joystick support by default, so we'll have to set that up. Run `fuse-sdk`, then press **F1** for the menu. Go to Options > Peripherals > General. Press **SPACE** to enable Kempston joystick support, then press **ENTER**.

Press **F1**, then Navigate to Options > Peripherals > Joysticks and make sure both Joystick 1 and Joystick 2 are set to Kempston. If not, press **ENTER**, press **ENTER** again to open the Joystick type options, navigate to Kempston on the list, and press **ENTER** again.

Note that some games may default to using Joystick 2, so you'll want to configure both, even if you only have a single stick connected.

When you're happy with your settings, open the Options menu and select Save.

10 Game configuration

Although many Spectrum games support joysticks, you'll often have to enable support for these. L'Abbaye des Morts enables joystick support by default, but its menus provide a good example of what to look for.

Load the game and then press **C** on the keyboard to access the control configuration. Pressing **1** here enables and disables Kempston joystick support. In other titles, you may need to explicitly choose to use your joystick to control the game if you want it to work.

11 Get game

The Spectrum's been a long-time homebrew favourite, with software continuing to come out for years past its original availability. There's been a resurgence in popularity of the platform with the release of a number of successors, most recently the ZX Spectrum Next.

As ever, the indie-friendly itch.io digital distribution platform is one of the best places to find both free and commercial releases for the Spectrum, and we've put together a list at **magpi.cc/zxspectrumgames**.

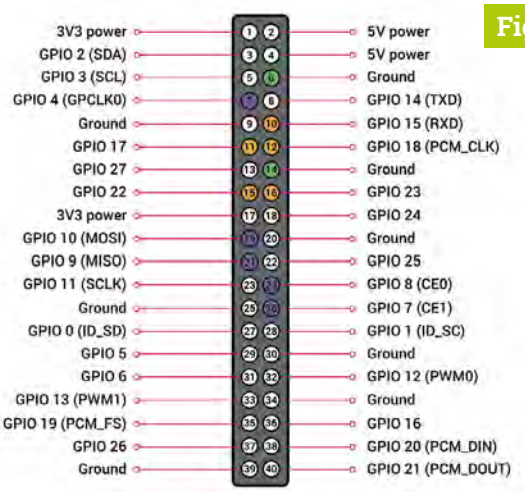


Figure 1

Figure 1 GPIO connection points for two single-button joysticks, corresponding to the 'GPIO connections' table (below). Joystick 1 is purple, joystick 2 is orange. Use your choice of ground pins for each controller: ground pins 6 and 14 are marked in green here

12 Boot to black

Finally, let's start FUSE on boot for that authentic Spectrum ambience. In **/home/pi/.config/autostart** create a text file called **fuse.desktop**. If the directory doesn't exist, create it. Add the following lines to your new file:

```
[Desktop Entry]
Type=Application
Name=FUSE
Exec=/usr/bin/fuse-sdl --full-screen
```

You can exit FUSE at any point to return to Raspberry Pi OS's familiar desktop.

Top Tip

Get in touch

Is retro gaming your hobby? Drop KG a message on Twitter **@KGOrphanides** if you have any early-2000s physical Linux game releases.

GPIO connections

GPIO connection table for two one-button joysticks. The db9_gpio_rpi driver uses these pins by default for single-button joysticks. You will also need a ground connection for each.

BUTTON	Joystick 1 GPIO PIN	Joystick 2 GPIO PIN
UP	GPIO 4	GPIO 15
DOWN	GPIO 7	GPIO 17
LEFT	GPIO 8	GPIO 18
RIGHT	GPIO 9	GPIO 22
FIRE	GPIO 10	GPIO 23

Commodore 64 Revamp

MAKER

Stephen Williams

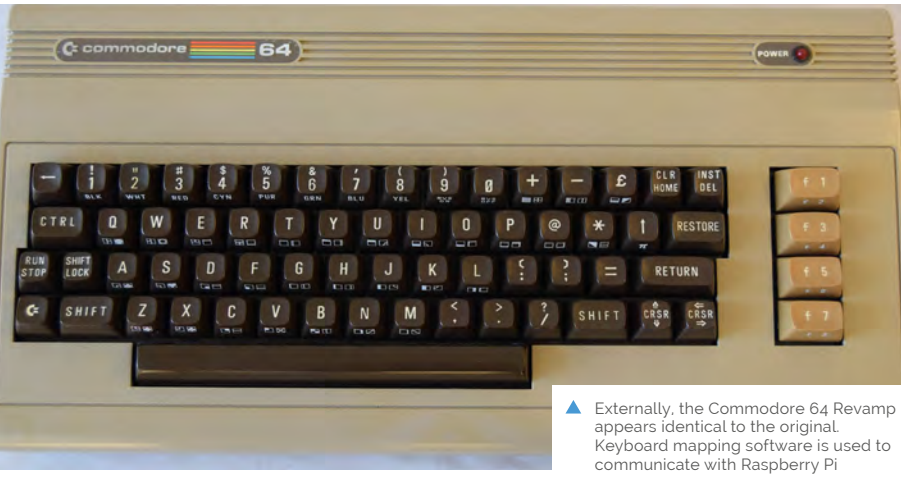
Stephen is an original 1980s retro-computer owner and a jack-of-all-trades who is happy to try new things. When not tinkering, he likes getting outdoors, exploring, and spending time with his family.

magpi.cc/c64revamp

Be like Stephen Williams and bring an old computer back to life. **David Crookes** takes a look

As many readers will know, Raspberry Pi can be turned into a brilliant, action-packed retro gaming arcade. Using operating systems like RetroPie, you can easily switch between emulators of many age-old home computers and consoles, and scores of makers have made use of this in various weird and wonderful ways.

In this instance, Stephen Williams has brought a broken Commodore 64 (C64) computer back to life. He's stripped out the original motherboard, replaced it with a Raspberry Pi computer, and used Lego® bricks to build the internal housing. It reminds us of Christian Simpson's fantastic Brixty Four project (**magpi.cc/brixtyfour**). But while that sought to create a new C64 case out of Lego, Stephen's project retains the original, iconic 'breadbin' plastic.



▲ Externally, the Commodore 64 Revamp appears identical to the original. Keyboard mapping software is used to communicate with Raspberry Pi

"I've played with Raspberry Pi since the computer first came out, making and discovering new things along the way," he tells us. "I had a truly broken Commodore 64 that I felt would benefit from a new lease of life. My project blends retro with a modern twist, and brings together some of the things I've liked to play with over the years – notably Lego, the C64, Raspberry Pi, and Arduino."

Building the project

Stephen turned to Lego for a practical reason. "I didn't have access to a 3D printer, and Lego has so many different pieces that it's really versatile to experiment with," he says. "I already had a Lego Raspberry Pi case, so it seemed to make sense to build around it. I wasn't sure what the fit would be like inside the C64, so I bought a mix of pieces from a market stall to see what would work."

He says it was important to build a solid base inside the C64 and position the USB, HDMI, and other ports in the right places. "I didn't want to modify the computer's case in any way and Lego helped me to do all of this." The rest of the build was rather straightforward and involved inserting a Raspberry Pi computer into the case so that the ports were accessible, inserting a microSD card with RetroPie installed on it, and connecting to the C64 keyboard.

To do this, Stephen used an Arduino Micro. "It provides the mechanism to get a fully working C64 keyboard for Raspberry Pi," he explains. "The basic idea is to scan the pin readings on the Arduino which are connected to the row and column pins on the C64's matrix keyboard. Using the Arduino

The cables are routed to the original holes in the C64 case and they're connected to Keystone inserts to provide accessible USB, HDMI, and network ports

Raspberry Pi sits within the Multicomp Pi-BLOX case which Stephen bought from The Pi Hut

Using Lego inside the original Commodore 64 case enabled Stephen to keep the components firmly in place

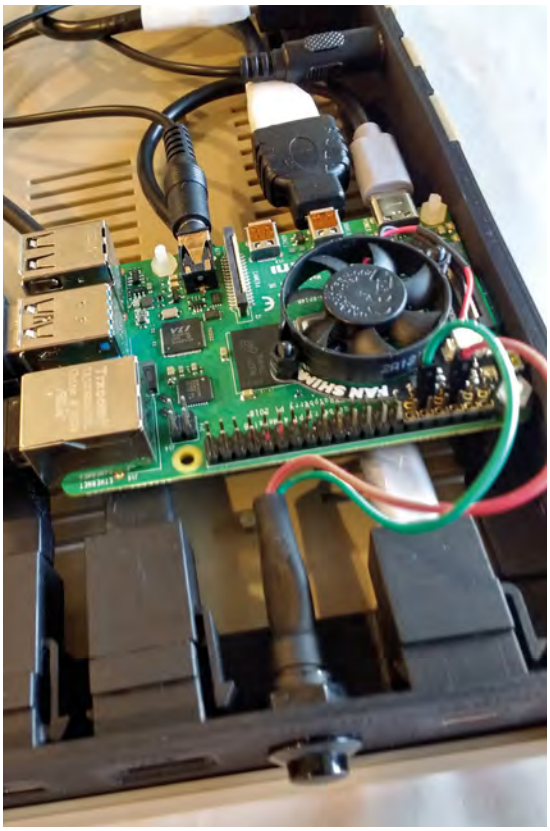
> Lego has so many different pieces, so it's really versatile to experiment with

software libraries, the row and column pins are scanned, and the mapped keystrokes are sent to the computer connected to the Arduino via USB."

Pulling it apart

Since creating this project, Stephen has acquired a 3D printer. As such, he's been replacing the Lego using printed parts, again for practical reasons. "When Raspberry Pi 4 came out, I wanted to use it but because I needed to install a fan, I couldn't use the Lego Pi case any more," he says. "The 3D build means I've been able to get closer to the original Commodore 64 regarding the location of the power socket and switch."

Even so, he's not always entirely faithful to the C64. "Since RetroPie brings many emulators into one place, it's been a bit surreal playing a Spectrum game with the C64 sitting in front of me, but I've become used to it. RetroPie is also easy to extend to include things such as homebrew programs, media players, and bespoke themes. It's been fun to dabble with these too."



The 3D-printed build, for comparison. Different cable requirements are also needed for Raspberry Pi 4 setups due to it using micro HDMI and USB-C ports

Quick FACTS

- No coding or electronic skills are needed
- The original C64 case is not modified either
- It's a great way to revive a broken C64
- You can even use USB joysticks
- Try applying the build to VIC-20 and C16 computers

Turn Raspberry Pi 400 into a legal C64 emulator

UK copyright law puts C64 emulation enthusiasts in a difficult position. This new VICE emulator fork replaces system files of unknown origin with official versions

MAKER

K.G. Orphanides

K.G. makes, writes about, and helps to preserve unusual gaming software and hardware.

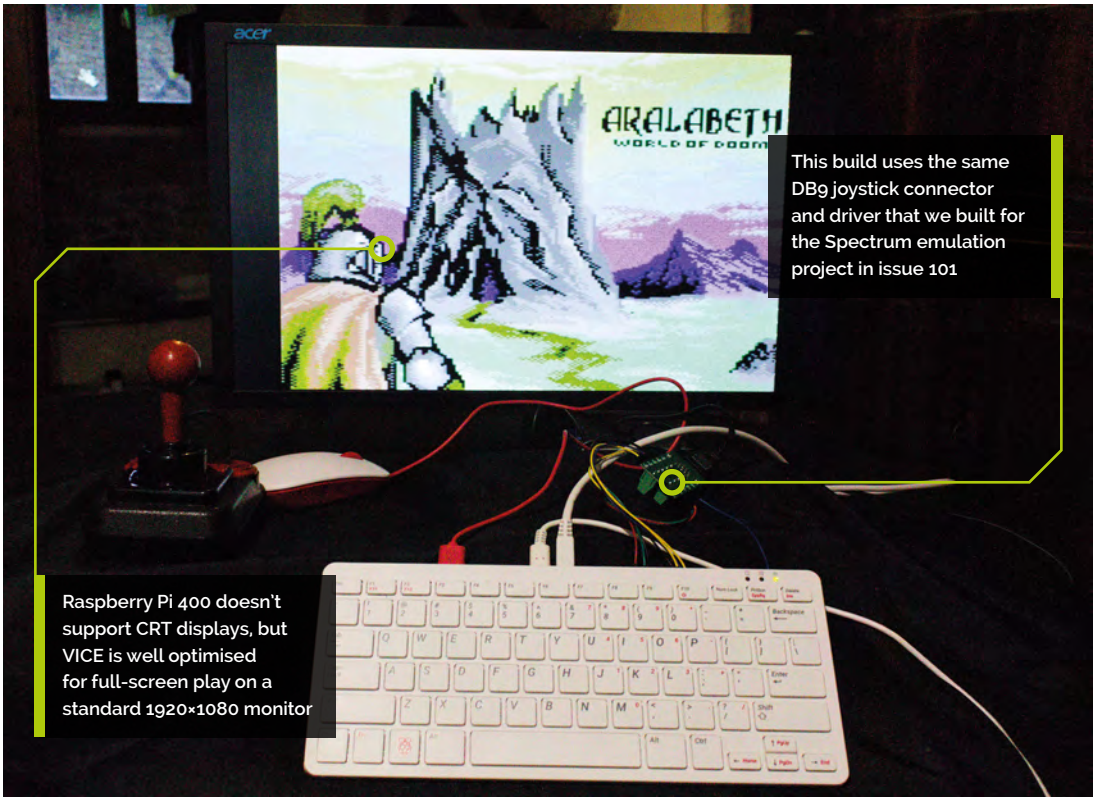
@KGOrphanides

If you're a fan of Commodore's classic 8-bit computers, you'll have noticed that the VICE emulator is nowhere to be found in Raspberry Pi OS's software repositories. Unlike some emulators, which use clearly authorised or reverse-engineered ROM images of the original computers' firmware, the ROMs included in VICE are ambiguously sourced, with little explicit licensing information.

That's a problem for anyone who wants to use VICE in the UK, as British copyright law is particularly stringent. Fortunately, almost all of Commodore's firmware is legally available for free via current licence holder Cloanto.

A fork of VICE 3.4, the most recent version of the emulator that runs smoothly on Raspberry Pi, has been created. This fork is stripped of any ROMs whose licensing status is unclear. It ships with a script to integrate Cloanto's ROMs.

This tutorial stands alone to give you a copyright-compliant C64 emulator on Raspberry Pi 400, but can also build on last month's Spectrum emulation project, using the same GPIO joystick controller setup.



You'll Need

- VICE 3.4 copyright compliant fork **magpi.cc/vice**
- C64 Forever **c64forever.com**

Warning!
Copyright

Only some C64 games can be legally downloaded. See our Legal ROMs page for more information. **magpi.cc/legalroms**

01 Preparation

If you followed last month's tutorial in its entirety, you'll probably want to undo the final step, which opens FUSE on boot. Delete or rename **/home/pi/.config/autostart/fuse.desktop**.

Next, we'll download Cloanto's legal Commodore ROM set, C64 Forever (**c64forever.com**). It's only available as a Windows MSI file. If you have a Windows PC, simply install it.

If you have an x86-based Linux or macOS system, Wine 5.0 and above (**winehq.org**) includes full MSI file support, so you can similarly install that, run the installer from your file manager, locate your Wine prefix directory (**~/.wine** by default), and you're good to go. Users of older versions can open a command terminal and run:

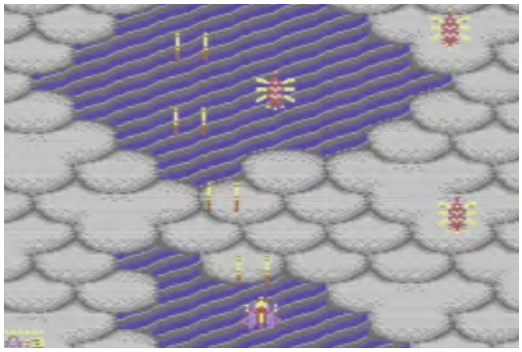
```
wine msiexec -i c64forever8.msi
```

If Raspberry Pi is your only computer, you can use Box86 (**magpi.cc/box86**) to run an x86 version of Wine; that's an extensive emulation project in its own right, but the TwisterOS image (**magpi.cc/twisteros**) comes loaded with Box86 and Wine to make this a little easier.

02 Locate your system ROMs

C64 Forever Express is free and includes all the ROMs you'll need for most Commodore 8-bit systems, except the PET and the C64 Direct-to-TV released by Ironstone. It also lacks Creative Micro Designs' SuperCPU ROM, but our fork uses an SCPU64-compatible ROM created by the VICE team.

After installation, you'll find the ROMs in **/users/Public/Documents/CBM Files/Shared/rom** of your Windows system drive. Copy this directory over to Raspberry Pi.



▲ With pin-sharp controls, Sarah Jane Avory's award-winning Zeta Wing is a stunning example of the new wave of commercial C64 games

▲ VICE's menu is hugely comprehensive, but you'll mostly interact with its Machine, Drive, and Video settings

A full C64 Forever Plus licence ($15 / £11 from **c64forever.com**) includes extra features for Windows and Wine users, and it's also the obvious choice if you wish to support Cloanto financially.

Both versions include a collection of around 100 games that Cloanto licensed from its developers and publishers, including Jack the Nipper, Stormlord, and Auf Wiedersehen Monty. Registered users will find the games on their Windows system drive at **/users/Public/Documents/CBM Files/Games** – copy that over, too, for later use.

03 Get ready to build VICE

We'll be using a version of VICE 3.4 that includes no copyrighted firmware ROMs. It won't build or work without ROMs, so we'll replace these using a script that ships with this fork. In a Terminal, enter the following two commands:

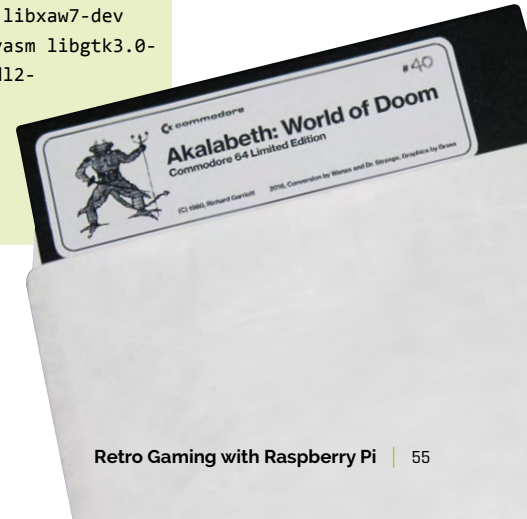
```
sudo apt install autoconf automake build-
essential byacc dos2unix flex libavcodec-
dev libavformat-dev libgtk2.0-cil-dev
libgtkglext1-dev libmp3lame-dev libmpg123-
dev libpcap-dev libpulse-dev libreadline-
dev libswscale-dev libvte-dev libxaw7-dev
subversion texi2html texinfo yasm libgtk3.0-
cil-dev xa65 libsdl2-dev libsdl2-
image-dev

git clone https://gitlab.
com/mighty-owlbear/vice-3-4-
copyright-compliant-uk.git
```

Now copy the **rom** directory from C64 Forever into your unpacked **vice-3-4-copyright-compliant-uk** directory.

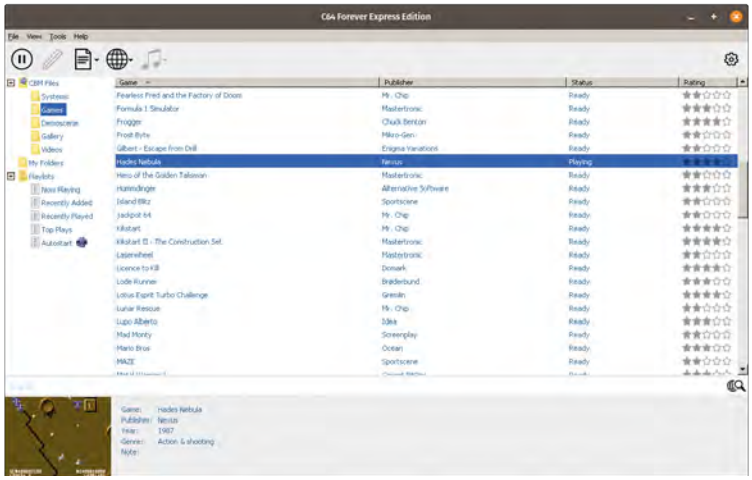
Top Tip

RTFM

The VICE manual is a chunky but vital reference. Read it online at **magpi.cc/vicemanual**.



▼ Physical games are still published for the C64, like this 5.25-inch floppy disk release of Dungeoneer's Akalabeth port, but you need extra hardware to use them



▲ C64 Forever is a native Windows program, but both its free Express and paid-for Plus editions include fully licensed Commodore ROMs for use with emulators

04 Build and install

The **copyrom** script supplied with this version of VICE looks for official ROM files in a directory called **rom** and copies and renames them as expected by VICE. Not all system ROMs are available; where this is the case, the script creates dummy files. If further ROMs are officially released, these can be replaced in an installed version by manually overwriting the files in **usr/local/lib/vice**.

```
  cd vice-3-4-copyright-compliant-uk
  ./copyrom.sh
  ./autogen.sh
  ./configure --enable-sdlui2 --without-
oss --enable-ethernet --disable-catweasel
--without-pulse --enable-x64
  make -j $(nproc)
  sudo make install
```

Note that you have to build VICE `--without-pulse`, as above, for working audio. If you'd rather install VICE to a user directory, you can add `--prefix=/home/pi/viceinstall_3.4_clean` and run `make install` as a standard user, instead of using sudo.

05 Test VICE

Although VICE includes several different emulators, we're primarily interested in the C64. Two different C64 emulators are supplied: x64sc is more accurate, while the older x64 emulator – which we specifically built using the `--enable-x64` parameter above – is less CPU intensive, making it the best choice for Raspberry Pi.

In the Terminal, we'll create a directory for your C64 game collection, download Kryoflux's authorised C64 release of Cinemaware's Defender of the Crown, and use it to test the emulator.

```
  cd
  mkdir C64 && cd C64
  wget http://www.kryoflux.com/download/
DEFENDEROFTHECROWN.zip
  x64
```

VICE should open with a blue window and an invitation to press **F12**. Do so, and marvel at the sheer number of configurable options available. Fortunately, we'll only need a few of them. But let's start by loading that game…

Press **ENTER** on 'Autostart image', browse to **DEFENDEROFTHECROWN.zip**, press **ENTER**, go down to the PRG file called **"!"V-MAX!** and press **ENTER** to load it.

06 Configure inputs

The GPIO joystick setup we built works perfectly with VICE 3.4, as will any USB joystick or joypad, but you'll notice that it initially only works in the menus.

Press **F12** to open the menu. Go to Machine settings > Joystick settings > Joystick device 2. Press **ENTER** to select it, scroll down to Joystick at the bottom, and press **ENTER** again.

Press left on your keyboard or joystick to go back up one level to the Joystick settings menu. Go down to 'Joystick 2 mapping' and select it. Select each of Up, Down, Left, Right, and Fire in turn, and tap the joystick button you want within the five-second configuration period to assign it.

If you accidentally assign the wrong button to one of these,

▶ Cinemaware's Defender of the Crown is one of a couple of games licensed for exclusive distribution by Kryoflux as a demonstration of its imaging system's handling of copy-protected C64 disks

Published by Protovision, Lasse Öörni's MW Ultra is one of many modern C64 games to get both physical and digital releases

use the keyboard to navigate back and reset it. Press **ESC** to return to your game.

Note that most, but not all, single-player C64 games default to using Joystick 2.

07 Tweaking VICE

VICE x64 works smoothly on Raspberry Pi 400, but you'll want to delve into its settings menu to get the most out of it.

First, go to Video settings > Size settings and enable Fullscreen. To provide a bit more screen real estate, go to Video settings > VICII border mode and select None. Test your setup again with your favourite game or demo.

Getting rid of the border on a 1920×1080 display may result in some graphical overspill to the right-hand side on some programs that use unusual parallax scrolling tricks, but it's a very minor and generally invisible issue.

Once satisfied with your configuration, go to Settings management > Save current settings.

08 Homebrew haven

VICE supports vanilla PRG program files, D64 disk images, CRT cartridge images, and T64 and TAP tape images, among others. As usual in these tutorials, we largely recommend modern C64 software, which is often distributed digitally, as the easiest way to get top-notch free and commercial games for your emulated retro system. You can find an itch.io C64 collection at **magpi.cc/itchc64**.

Specialist publishers such as Psytronik (**psytronik.itch.io**), Double Sided Games (**doublesidedgames.com**), Protovision (**protovision.itch.io**), and BitmapSoft (**bitmapsoft.co.uk**) also continue to put out spectacular commercial games for the C64.

Meanwhile, passion projects have seen authorised ports of games from other systems, such as Dungeoneer's C64 version of Richard 'Lord British' Garriott's Akalabeth (**magpi.cc/akalabeth**) and Double Sided's release of L'Abbaye Des Morts (**magpi.cc/desmorts**).

09 Using C64 Forever's RP9 files

If you wish to use the games that came with C64 Forever, you'll have to rename them from the default RP9 extension. On Raspberry Pi OS, you can use the rename command-line tool for this. Drop the entire zip files into your C64 games directory, open a Terminal in the directory, and enter these commands:

```
sudo apt install rename unzip
rename -v 's/.rp9/.zip/' *.rp9
```

Once renamed, you'll find that the zip files contain either D64 disk images or T64 tape images. Extracting them makes it easier to distinguish one format from the other.

```
unzip -o \*.zip
rm *xml *.txt *.png *.zip
```

Although many of the game files supplied are cracked versions, Cloanto makes it clear that licences to distribute the titles have been obtained from their original developers. Licensing details are provided in the Cloanto EULA.

Top Tip

Demoscene conversion

Demos show off the C64's sound and graphics capabilities. Try The Elder Scrollers from **magpi.cc/c64scrolls.**

Raspberry Pi Amiga 600

Billy Nesteroulis has created an Amiga computer for the modern user, as **David Crookes** explains

MAKER

Billy Nesteroulis (DJ Nest)

Billy is an Amiga musician and a member of the Vintage Computers Society of Athens. His team specialises in 3D prints and he loves to experiment with Raspberry Pi.

magpi.cc/djnest

Even though the Amiga range of computers ceased production in 1996 following a successful eleven-year run, many users remain determined to keep its memory alive.

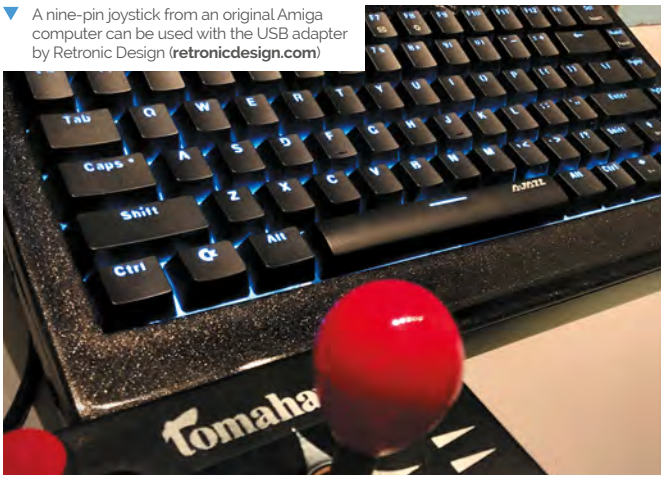
Not only has a new magazine recently emerged (Amiga Addict), but Commodore's machine has resurfaced in various guises over the years. Its operating system, AmigaOS, continues to be updated, and there was even an A500 MINI released recently with classic games installed.

Such news excites Amiga fans. "The price of used Amigas has skyrocketed over the last five years and it's not an easy task to preserve an old computer," explains Billy Nesteroulis, aka DJ Nest. "If you own an old Amiga, it will eventually break: their electrolytic capacitors tend to leak. You'll need a new power supply, and some kind of memory expansion is ideal."

With a Raspberry Pi computer, however, such costs can be significantly lowered. As Billy has shown, it's possible to build an Amiga 600 from scratch with a Raspberry Pi 4 as the main unit.

▶ With Amibian and Amiberry as the main emulator, you can emulate any Amiga model you like

▼ A nine-pin joystick from an original Amiga computer can be used with the USB adapter by Retronic Design (**retronicdesign.com**)



"Raspberry Pi can emulate an Amiga with AmigaOS and you can use it to play games and software made for the machine," he continues.

Stars in their eyes

Certainly, Raspberry Pi has proven to be the perfect platform for Amiga emulation. "Dimitris Panokostas has done a remarkable job creating the Amiberry emulator and because Raspberry Pi hardware is small, it can fit easily almost everywhere," Billy says.

In this instance, the single-board computer has been fitted inside a full-size, 3D-printed replica of an Amiga 600 case, allowing use of its USB ports and wireless LAN. A specially designed keyboard that was originally designed as a replacement for ageing Amiga machines is connected and modern adapters will allow use of the nine-pin joysticks of old for added authenticity.

"The Cherry MX keyboard is illuminated and it was designed to fit the case that I 3D-printed," Billy explains. "The joystick adapter is plug-and-play with no drivers needed and you can also use Amiga CD32 joypads with their eight buttons." Other parts include a micro HDMI extender, SD card extender, power supply unit, USB extenders, a power switch, and LAN extender.

Quick FACTS

- The project costs around $250 in total
- It can emulate Amiga 500s to Amiga 4000s
- But the case is modelled on an Amiga 600
- You can plug it into a modern monitor
- Amiga novices could use the PiMIGA emulator

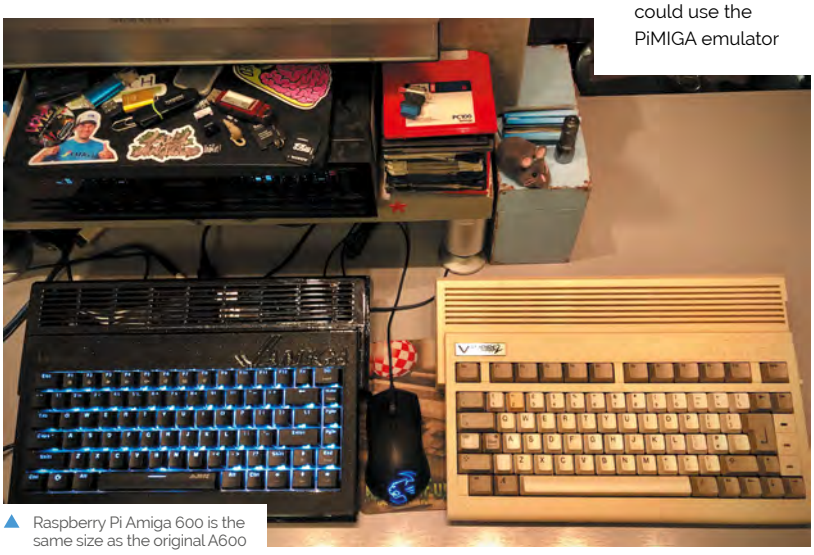
To ensure everything runs smoothly, Billy uses the Amibian distro ("the most complete experience of the classic Amiga environment"). He also likes that – in exchange for a small donation – he can use the Amibian 1.5 Extended Edition made by Gunnar Kristjánsson. "The Extended Edition includes Raspbian Buster V10 OS with the look and feel of the Amiga OS 4," Billy says. "It has a modern browser, the VLC media player, and the Qmmp audio player. You can even use LibreOffice Writer."

> Raspberry Pi can emulate an Amiga with AmigaOS and you can use it to play games and software

A modern touch

Amibian also allows users to update software and Amiga emulators through its configuration tool. All of which has meant Billy's set up expands the potential of the machine, beyond matching the real A600. "It's allowed me to bond classic computing with modern computing," he says.

Indeed, Raspberry Pi 600 gives the same feeling and experience of the A600, but with the modern touch of the Raspberry Pi hardware. "It has the required juice to run specific software such the classic pixel-art package Deluxe Paint, games play without issues, and you can build your own system and adapt it to your needs," Billy says. "For many people, it's the best Amiga solution in 2021."



▲ Raspberry Pi Amiga 600 is the same size as the original A600

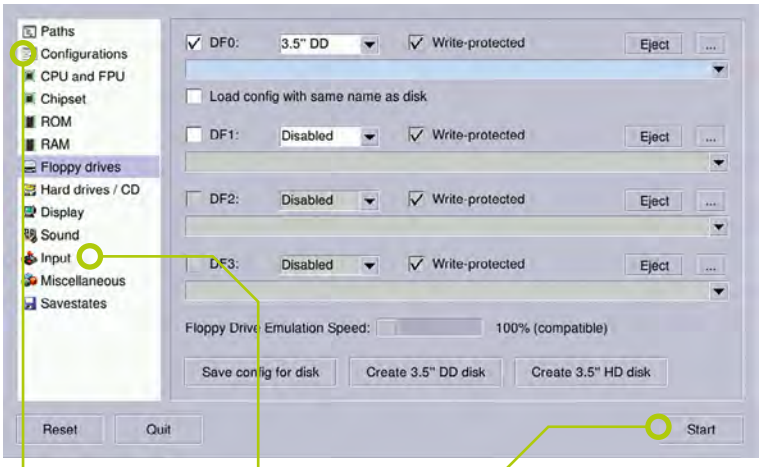
Turn Raspberry Pi into an Amiga

Recapture the glory days of 16-bit computing by turning your Raspberry Pi into a faithful Amiga emulator

You'll Need

- microSD card
- USB stick
- Wired Xbox 360 controller
- Amiga Kickstart ROMs **amigaforever.com**
- Amibian **bit.ly/Amibian**

The Commodore Amiga's top-notch sound and graphics made it one of the most desirable home computers of the 1980s and early 1990s, at a time when your average IBM PC was still plodding along with EGA graphics and an internal beeper. Amiga games from the era have aged incredibly well, and look and play brilliantly on everything from a portable display to a widescreen TV. We'll take you through turning your Raspberry Pi into a perfect modern-day Amiga emulator. You'll need a Windows, macOS, or Linux desktop operating system to copy the Amibian Linux distribution to your SD card and unpack the Kickstart ROMs required to make it work smoothly.



You can load and create emulated hardware configurations for specific Amiga computers

Game controllers, mouse, and keyboard configurations can be selected and tweaked in Input

Once you've set up your emulated hardware and firmware config, just mount a floppy disk image and click Start

Start by downloading the Amibian distro. Format a microSD card, decompress the Amibian RAR file, and use Win32DiskImager or Linux's dd command to copy the IMG file to the card. A 4GB card should be plenty, as Amibian only occupies around 300MB. Slot the microSD card into your Raspberry Pi and power up. It'll boot directly into the UAE4ARM emulator, but there's some extra configuration to do before we start playing. Quit UAE4ARM to get to the command line and run:

```
raspi-config
```

Select Expand Filesystem, which will give you access to the entirety of the SD card's capacity for storage, then Exit and select Yes to reboot.

If your Raspberry Pi won't output sound via HDMI properly, enter this at the command line:

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

Make sure the following lines are present and aren't commented out with a preceding hash (#):

```
hdmi_drive=1
hdmi_force_hotplug=1
hdmi_force_edid_audio=1
```

Save and return to raspi-config:

```
raspi-config
```

Select Advanced Options > Audio > Force HDMI and then reboot.

Kickstart me up

To run Amiga programs, you'll need a Kickstart ROM – firmware from the original computers. UAE4ARM comes with the open-source AROS ROM, which can run only some Amiga programs, so we

recommend using genuine Amiga Kickstarts for reliability. The Kickstart ROMs and Workbench GUI are still being maintained, thanks to Italian firm Cloanto. Amiga Forever Plus Edition, priced at €29.95, gets you a complete, legal set of Kickstarts for every Amiga computer and console. As there's no Raspberry Pi edition, yet, you'll currently have to install Amiga Forever on a Windows PC or Wine and copy the files onto a USB stick.

There are other ways of obtaining Kickstart ROMs, but most are legal grey areas. You can extract them from an Amiga using a tool such as TransRom or find them on abandonware sites, but we strongly recommend supporting Cloanto's continued development of Amiga Forever.

Classic Amiga software is even easier to find. You'll get 50 games with Amiga Forever Plus, while some major publishers have made the Amiga versions of their games available for free.

One true path

As Amibian doesn't include a window manager, it's easiest to download and copy everything to a USB stick using your operating system of choice. Helpfully, UAE4ARM can read Amiga ADF floppy images even if they're in a ZIP file.

We advise copying everything to your microSD card. Start Raspberry Pi, exit UAE4ARM, and run:

```
mc
```

Copy your game files from **/media/usb** to **/root/amiga/floppys**, and your Kickstart ROMs, including a Cloanto **rom.key** file if you have one, to **/root/amiga/kickstarts**. Quit Midnight Commander and reboot:

```
reboot
```

In the latest version 1.313 of Amibian, two different versions of UAE4ARM are supplied. If you plan on using two Xbox 360 controllers, button mapping on controller two works best using the 'old' version, although the 'new' edition generally provides more options. To switch between the two, at the command line type either `rpiold` or `rpinew`. The following configuration instructions work with both versions.

Configure UAE4ARM

First, go to the Paths tab and click Rescan ROMs so UAE4ARM knows where to find everything. The Configurations tab lets you select from several preset hardware emulations, with the default being an A1200 – just select and Load your chosen computer. You can tweak your virtual hardware in the CPU and FPU, Chipset, and RAM tabs.

Your configuration selection doesn't always set the relevant Kickstart ROM for you, so check the ROM tab, where you can choose Kickstarts from a pull-down menu. Note that many games require a specific ROM or hardware configuration to work properly, depending on which system they were originally released for.

To run most software, you'll need the Floppy drives tab. Just press the … icon next to drive DF0's Eject button, select the desired disk image, and click Start. By default only drive DF0 is active, and most titles expect this configuration. To swap disks when prompted, press **F12**, eject the disk image in DF0, select the disk image you're asked for, and click Resume.

F12 will always pause and return you to UAE4ARM's main interface, so you can create a save state – a stored image of your progress in a game – or give up and load something new. The Reset, Quit, and Start/Resume buttons are always visible in UAE4ARM's GUI. Reset completely reboots your emulation and Resume returns you to your current game.

UAE4ARM automatically detects Xbox controllers. You can use two controllers for multiplayer gaming – if the second is unresponsive, you may need to press **F11** to disable your mouse and switch control to the pad. If you're running the 'new' version of the emulator, first select your controllers from the pull-down Port0 and Port1 menus in the Input settings.

Now you've got your Amiga emulator up and running, there's plenty of scope to build on the project, from setting up virtual hard disks to install Workbench and other software onto, to creating floppy images from your own original Amiga disks and using Raspberry Pi's GPIO to connect a classic 1980s joystick.

Top Tip

Publisher-approved game downloads

Amigaland
amigaland.de

Ami Sector One
magpi.cc/2dDLEIL

Gremlin Graphics World
magpi.cc/2dDKZ3S

Enjoy some lunchtime classic arcade action with this compact gaming system

Lunchbox Arcade Game

A gaming fan dumped his sandwiches to become a legend in his own lunch hour. **Rosie Hattersley** finds out how

MAKER

Rich Jones

Engineer Rich lives in north Wales and has built several PCs over the years. He recently began focusing on building arcade machines, hence his YouTube tag 'Arcade Dad'.

magpi.cc/arcadedad

A fondness for school lunches might be unusual, but we're sure Rich Jones isn't the only person nostalgic for a much-loved lunchbox. The engineer, from north Wales, found himself idly searching for lunchboxes emblazoned with his favourite game, Pac-Man, but baulked at the hefty price tags on eBay and Amazon.

He eventually chose one celebrating a different game, kicking off his Raspberry Pi-based Lunchbox Arcade Game project.

"I used to have a cool metal Pac-Man lunch box for school but trying to get a Pac-Man one is nearly impossible, and ones that do come up go for silly money on eBay," he explains. At more than £60 a tin, he couldn't bring himself to drill holes in an original 1980s version. Instead, Rich decided to modify a Gauntlet one.

Most of the parts, for what became a roughly £250 build, were sourced from Arcade World (**magpi.cc/arcadeworld**).

Lunch bunch

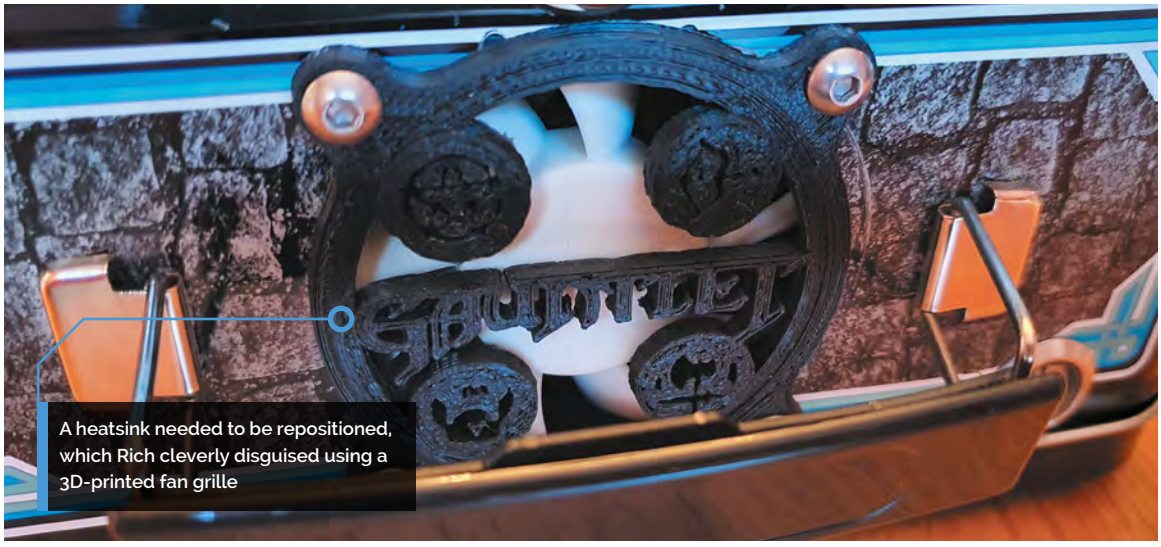
Rich had already built three Windows-based arcade machines having moved on from assembling his own PCs, plus one using Raspberry Pi. For his Lunchbox Arcades, Raspberry Pi was a shoo-in.

"So much power in such a small form factor makes Raspberry Pi a great choice for mini arcade machines," says Rich. "With all my machines the sound quality is important, so I've fitted the largest speakers the lunchbox could realistically support." He strengthened the tin all round using plywood in the base and fitted new rivets into the metal sides.

Next came the lid. He removed the original hinge, realising it wasn't strong enough to support

Inspired by his love of classics, Rich Jones came up with a retro arcade that fits inside a games-themed lunch tin

The Sanwa joystick has a detachable shaft that can be packed away inside the lunchbox arcade



A heatsink needed to be repositioned, which Rich cleverly disguised using a 3D-printed fan grille

Alert! Voltage Reducer

This project uses mains electricity with a voltage reducer. Be careful when working with electricity.

magpi.cc/electricalsafety

Quick **FACTS**

- The project took around a week to build
- For v2, Rich will be fitting a Raspberry Pi 4
- ...as well as adding battery power and charge circuits
- He's also on the lookout for a widescreen lunchbox
- ...for a 16:9 RetroPie console. of course

▲ Rich has embarked on a mini arcade collection having previously custom-made a cabinet for another of his self-built Raspberry Pi games machine

◀ A modified Sanwa joystick shaft with a quick-release system for easy removal

the LCD screen. "I was able to gain some height with new hinges. This allows the screen to fold inside the casing, giving a seamless look when shut," Rich says. "The joystick is removable to allow the lid to close. It's a modified Sanwa joystick shaft with a quick release system. The sprung release shaft comes off easily and can be stored in the back of the machine. The original latch holds the lid shut."

It wasn't all straightforward, though: "Airflow was also important, so a 60 mm fan forces air inside over Raspberry Pi and out of two slots cut in the control panel." With hindsight, says Rich, this could have gone on the rear rather than where the handle is. A Gauntlet fan grille he created on a 3D printer now covers it up a bit.

Power play

"The Lunchbox Arcade runs off a 12 V, 6 A power supply. A buck voltage reducer takes this down to 5 V for Raspberry Pi and the screen. "The buck voltage PCB will also look after a rechargeable battery, so I'm trying to source a 12 V battery that will fit in the case and also provide a good few hours playtime," says Rich.

▲ The sound quality was really important, so Rich bought a tin he liked but that wasn't so expensive that he had qualms about cutting it up to accommodate large speakers

▲ RetroPie and RetroArch games provide plenty of retro gaming options

> So much power in such a small form factor makes Raspberry Pi a great choice for mini arcade machines

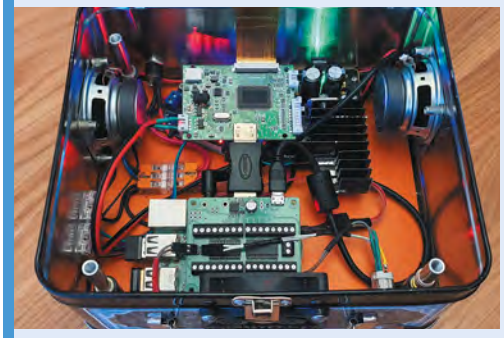
Although the project has the potential to be self-powered, he didn't want to compromise on the speakers. "The speakers and amp were the whole reason behind the 12 volt power supply. I didn't want some tiny speakers and a 5V amp." Despite having to reposition the heatsink to accommodate the amp, the audio setup proved worth the extra hassle. "The speakers sound really good as they resonate through the tin and have good bass, which surprised me," says Rich.

He thinks others might enjoy making something similar. Raspberry Pi is perfect for this size of machine. It has plenty of power, great visuals, and no slow-down in the games," he enthuses. If you're embarking on your own arcade project he advises, "Always think about maintenance and how you're going to access all the components in the future. The controls need to be easy to remove, so making up some sort of quick wiring connect that you can just unplug will save a lot of hassle in the future."

Box clever

Source a metal lunch box that's large enough to accommodate Raspberry Pi, speakers, and amp. You'll also need a joystick, ideally with a removable shaft so it can pack away inside your mini arcade box.

01 Connect Raspberry Pi to the LCD screen using an HDMI cable, then plug in an amp to the 3.5mm audio jack.



02 Use a USB keyboard converter as a control block to convert arcade buttons to keystrokes. Rich suggests sourcing these from **petrockblock.com.**

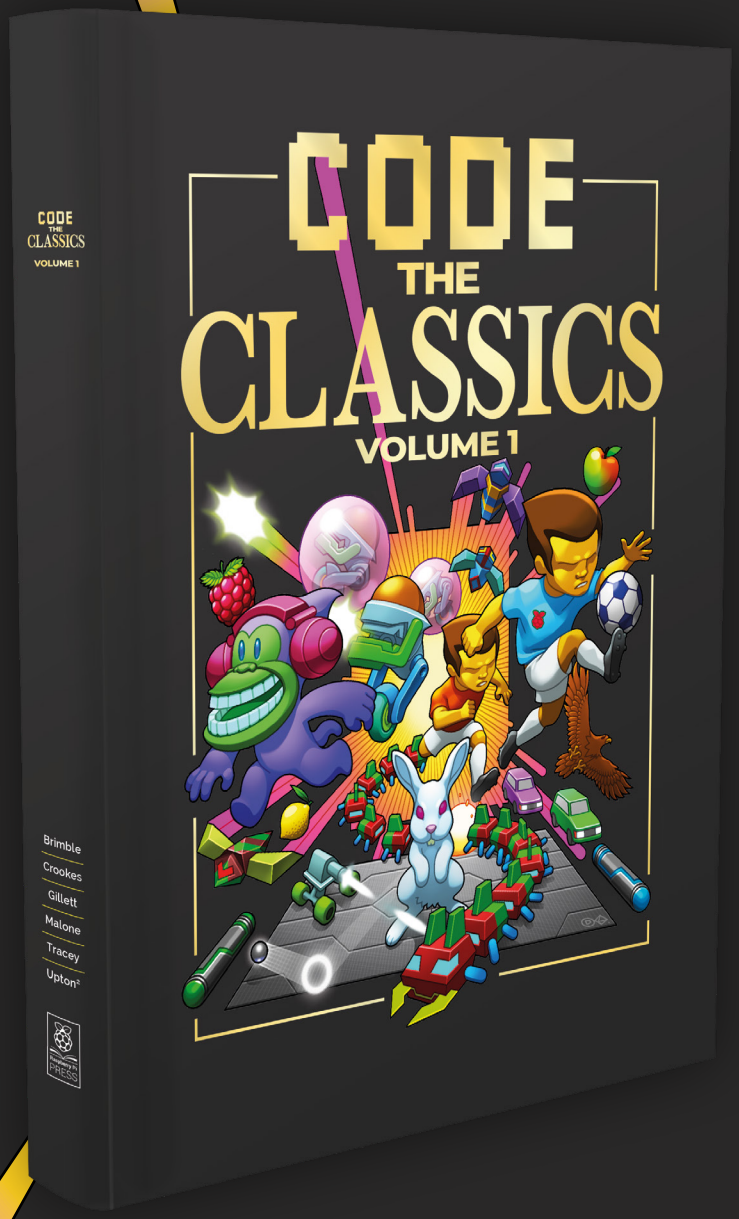
03 The control block enables the arcade to safely shut down via the power button on the front. Attach it to Raspberry Pi using the GPIO pins.

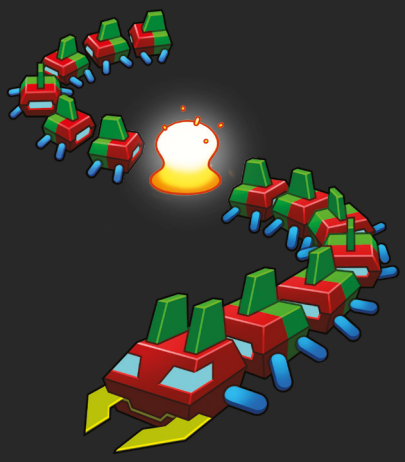
MAKE YOUR OWN GAMES

PROGRAM RETRO-STYLE GAMES WITH PYGAME ZERO

> Pygame Zero is a great choice for anyone who wants to start writing computer games


4000





- ***Get game design tips and tricks from the masters***
- *Explore the code listings and find out how they work*
- ***Download and play game examples by Eben Upton***
- *Learn how to code your own games with Pygame Zero*

This stunning 224-page hardback book not only tells the stories of some of the seminal video games of the 1970s and 1980s, but shows you how to create your own games inspired by them using Python and Pygame Zero, following examples programmed by Raspberry Pi founder Eben Upton.

Available now: **magpi.cc/store**

MAKE GAMES WITH RASPBERRY PI

Did you know that Raspberry Pi is a game-creation machine? There are many ways to write games and **Mark Vanstone** will show you a few to get you started

MAKER

Mark Vanstone

Educational games author from the 1990s, author of the ArcVenture series, disappeared into the corporate software wasteland. Rescued by Raspberry Pi!

magpi.cc/technovisual

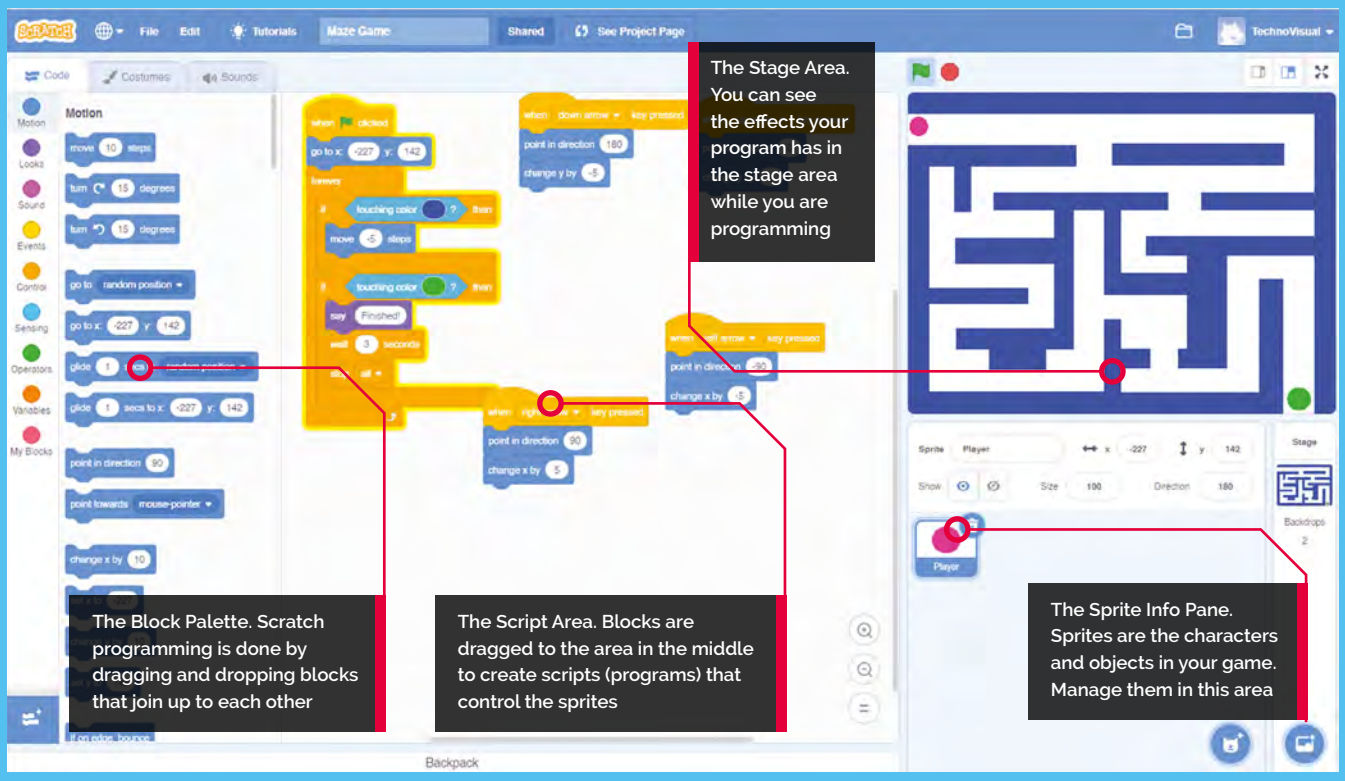
The UK computer games industry has grown and grown since its origin in the eighties; grown so much that it's on a par with the UK film industry. This trend is true in many other areas of the world. If you are learning how to write games, then Raspberry Pi is a great way to get your teeth into the subject. There are many ways to write games and get started quickly.

If you are just starting out and not ready for text-based coding, don't worry. There are block-based systems like Scratch where you can lay out your game graphics on the screen. You can code games using blocks that you drag and drop into place to create your program. Once you have mastered that, you may want to move on to text-based coding like Python and Pygame. If you aim to start a career in the games industry, you will find that these days game designers use both methods: visual block editing and text-based programming. One of the most popular game engines, Unreal Engine uses a block editor called Blueprint that is underpinned by libraries of C++ code.

Game programming is a great way to learn a wide range of techniques that are useful for other areas of programming. You don't need to start with advanced scripting but can easily get quick results with the tools in this article. Let's make some games!

MAKE GAMES WITH SCRATCH

Scratch is an ideal place to start making games, with a great online community of creators who share their games so that you can see how they were made



Top Tip

Using sounds

If you want to use sound in your project, you can go to the Sounds tab. You will find a cat meow sound to start you off.

Scratch is a block-based visual editing programming language. Instead of writing commands in text, you click and drag objects (known as 'sprites') and control them with block commands. It's designed to make object-oriented programming easy to understand, and is a great way to get to grips with coding concepts. Due to its visual nature, it's ideal for creating basic games and interactive stories.

There are several versions of Scratch that are compatible with all versions of Raspberry Pi although the latest version, Scratch 3 is recommended for Raspberry Pi 3 and Raspberry Pi 4.

01 Get Raspberry Pi ready

It's always a good idea to keep your system files up-to-date. You can either download a fresh install of Raspberry Pi OS to your system card using the instructions at **magpi.cc/imager** or from your Terminal window use the commands `sudo apt update` and then `sudo apt upgrade`. It is wise to always go through this procedure before installing anything new on your Raspberry Pi to make sure you have the latest version of all the system files. Of course, for any installs or updates, you will need a connection to the internet.

02 Install Scratch

There are three versions of Scratch and an online editor. You can install Scratch 3 by clicking on Menu > Preferences > Recommended Software. In the Programming section, you will see Scratch 3. Place a tick in the Install checkbox to the right and click Apply.

03 Your first Scratch

If you have not used Scratch before, you probably want to jump straight in and make something happen. With Scratch, you can do just that. You'll find Scratch in Menu > Programming > Scratch 3. You will see a cartoon cat on the right-hand side and a set of blue boxes on the left. Drag the **turn 15 degrees** block into the Script area in the middle (this is where you assemble your program). Click the **turn 15 degrees** block and the cat will rotate.

04 The green flag

Our rotation block is good, but what if you want something more? We can build a program of blocks by joining them together. Click Control in the sidebar and then drag and drop the **repeat 10 block** into the Script area. Then move your **turn 15 degrees** block so that it's inside the **repeat** block. Then click Events in the Blocks palette and drag the **when (green flag) clicked** block to sit on top of the **repeat** block. Now if you click the green flag at the top of the window, the program will run.

05 Customising your sprites

In Scratch, a sprite is a graphic on the screen that you are controlling. You can change what it looks like by loading in your graphics or editing the image. Select the Costumes tab at the top of the screen. From there you can use the painting tools to make a new image or alter the one that is already there. Try drawing some lines or filling in some shapes to see how it works. You can also write text. If you select items with the arrow tool, you can also change their colour.

06 Exploring the Scratch community

You will probably want to find out lots more about how to use Scratch, and there are lots of tutorials if you select the Tutorials section in the top menu bar of the desktop or online version. You can also get lots more help from the main Scratch web page at **scratch.mit.edu/ideas**. If you look at the Explore section on the website, you will be able to find lots of projects that other people have made and if you find one you like, you can see how they did it by selecting the 'See inside' button at the top-right corner.

Or you can try PICO-8

PICO-8 is a fantasy console for making, sharing, and playing tiny games and other computer programs. It feels like a regular console and runs on a variety of platforms. It has a suite of cartridge creation tools and an online cartridge browser called SPLORE. The programs are distributed in the form of a PNG file and each program has a memory limit of 32kB, so it is like programming a retro-style, 8-bit computer. The PICO-8 development system costs £11 / $15 and can be downloaded from **magpi.cc/pico8**.

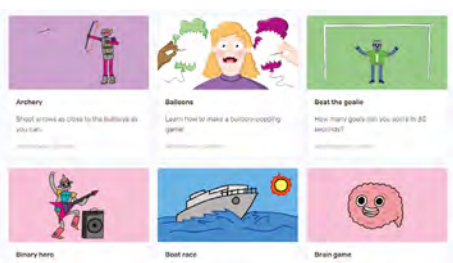
Top Tip

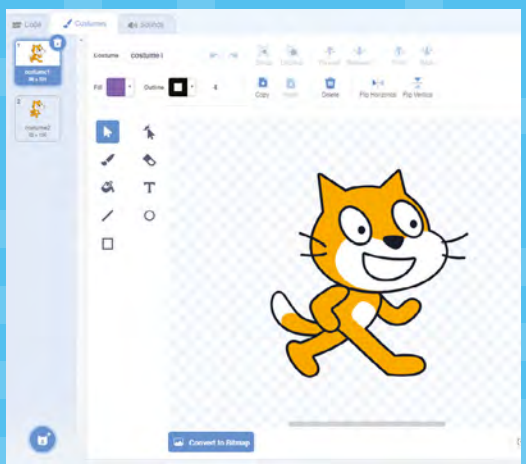
Loading and saving

You can load/save Scratch projects to your Raspberry Pi. The online version can save projects to the Scratch server if you log in.

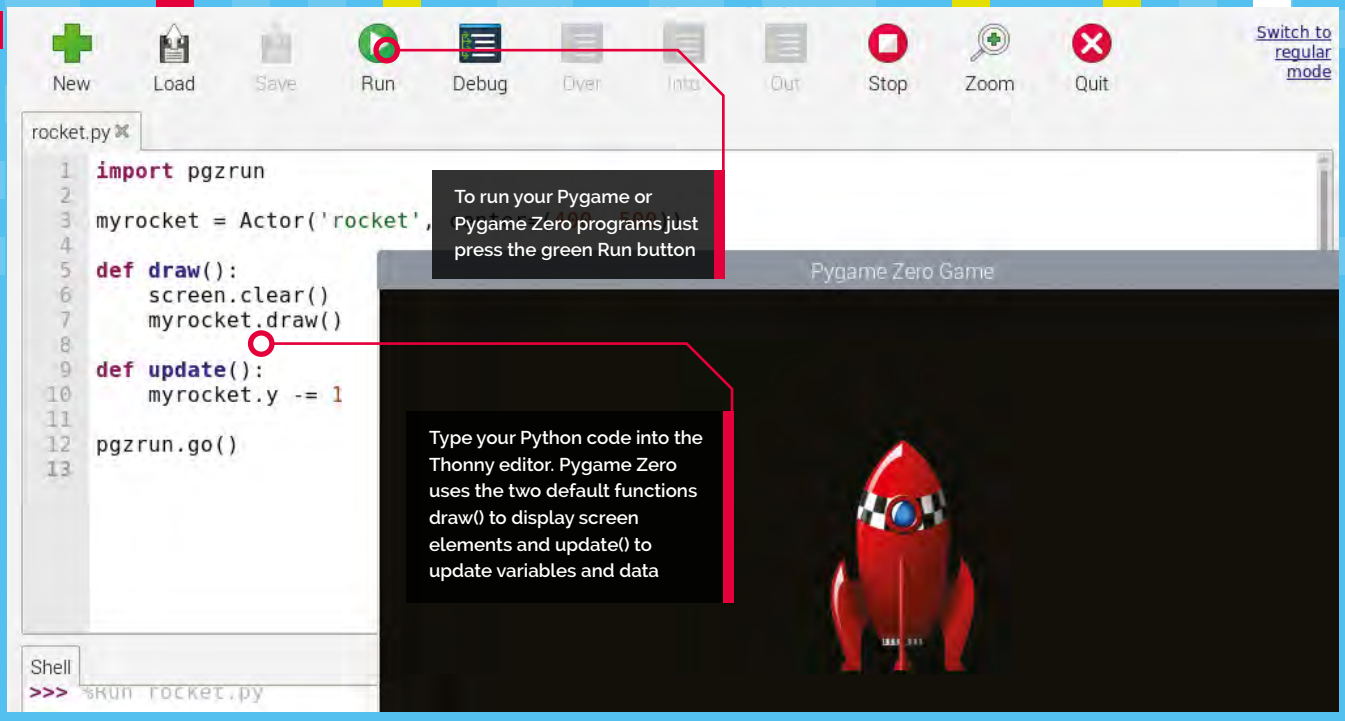
RASPBERRY PI SCRATCH PROJECTS

Go to the projects section of Raspberry Pi's website, **magpi.cc/projects**. In the 'Find a Project' section, select Games from the Topic drop-down, and Scratch from the Software drop-down. You will be shown a selection of game projects for Scratch. Each of these projects is laid out as a step-by-step tutorial to help you build the game. There are lots of different game projects available, so you shouldn't run out!





The costume editor allows you to edit and design graphics for your games



CODE PYTHON GAMES WITH PYGAME

Raspberry Pi can be used to make some super games and Pygame gives you a great head start

Top Tip

Mix and Match

Even if you start your program with Pygame Zero, if you need a function from Pygame you can include parts of the Pygame module too!

One of the best ways to get started with text-based programming on your Raspberry Pi is to jump straight into Pygame or Pygame Zero. These are both available with the Python programming language and all three are already installed by default with Raspberry Pi OS. If you are not familiar with Python, you can get it running from the Programming menu by selecting the Thonny Python IDE. This will open up an editor to use Python 3. Python is easy -to learn and read, and this article will show you how to use it with Pygame and Pygame Zero.

01 First Pygame Zero

Pygame Zero was designed to require as little code as possible to get a game running. If you launch the Thonny Editor (IDE) and type `import pgzrun` to load the Pygame Zero module and then after that, write `pgzrun.go()` to start the game, you can then save the file and run it (with the green play button). If you have typed the code correctly, you will see a black window appear titled 'Pygame Zero Game'. You have written your first Pygame Zero game! It's not a great game yet but that's all you need to get the game engine running.

02 More than zero

Let's get a graphic moving on the screen. You will need to find a suitable image to use, perhaps a spaceship or little green man. Have a look at the 'Graphics Resources' section near the end of this feature about where to find graphics. A PNG is best; you can find our rocket at **magpi.cc/rocketart**. Now make a subdirectory in the same place as you saved your Python file and call it **images** and put your graphic file inside that directory. Now load that graphic into an Actor object in your code. Name your graphic file **rocket.png** (you must keep to lower-case letters) and load it by typing `myrocket = Actor('rocket', center=(400, 500))`.

03 Seeing the rocket

Now to get our rocket to display on the screen, we need to add some code to draw it. We do this with a `draw()` function, so type `def draw():` and press **RETURN**, then type `myrocket.draw()`. Then, to make the rocket move up the screen, we need to add an `update()` function by typing `def update():` and underneath type `myrocket.y -=1`. If we save and run this program, we should see the rocket moving up

the screen. If you don't, check the **rocket.py** code to see what you have done differently. It may have some drawing left behind, so add `screen.clear()` at the beginning of the `draw()` function. If all is well, you have the start of your Pygame Zero game.

ROCKET.PY

Language: Python

```
import pgzrun

myrocket = Actor('rocket', center=(400, 500))

def draw():
    screen.clear()
    myrocket.draw()

def update():
    myrocket.y -= 1

pgzrun.go()
```

04 Moving on to Pygame

Pygame Zero makes it very quick and easy to get games working on your Raspberry Pi, but if you want more flexibility you may find that Pygame is what you require. You will need to write a bit more code, but you will be able to access some functions like using game controllers. To start a Pygame program, you will need to import the pygame module using `import pygame` and then after that, make a call to `pygame.init()`. This starts the game engine off, but we won't see anything happen if we run it.

05 Making a screen

We make a screen for our game by calling a function called `pygame.display.set_mode()` and give it the width and height that we want the screen to be. Once that is set up, we will need to start a loop (in this case a `while` loop) to check that the program is still running – and in the loop we blank the screen, draw our graphics on an invisible buffered screen, and then flip the screen from the buffer to the visible screen. All this keeps happening until the user exits the program by using the window close icon. Have a look at **pgtest.py** to see how all this is done with Pygame.

Lots of Pygame game developers share their creations online like this game called Dynamite

PGTEST.PY

Language: Python

```
import pygame
pygame.init()

screen = pygame.display.set_mode([400, 400])
running = True

while running:
    # Get events from the user
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False
    # Fill the screen
    screen.fill((0, 0, 0))
    # Draw a red circle at 200,200 with
radius of 50
    pygame.draw.circle(screen, (255, 0, 0),
(200, 200), 50)
    # Switch from buffered screen to visible
    pygame.display.flip()

# Quit the program
pygame.quit()
```

06 Taking it further

Both of these examples are very simple, just to get you started, but there are lots of amazing games that can be made with Pygame and Pygame Zero. Over the years, *The MagPi* magazine has featured many tutorials about making games in Python and has even produced three books dedicated to teaching Python Games by example: *Retro Gaming with Raspberry Pi* (**magpi.cc/retrogaming**) and *Code the Classics – Volume 1* (**magpi.cc/ctc1**) which have many Pygame Zero example games, and the other is called *Essentials – Make Games with Python* (**magpi.cc/essentialgames**) which takes you through game creation with Pygame.

Top Tip

Watch your naming

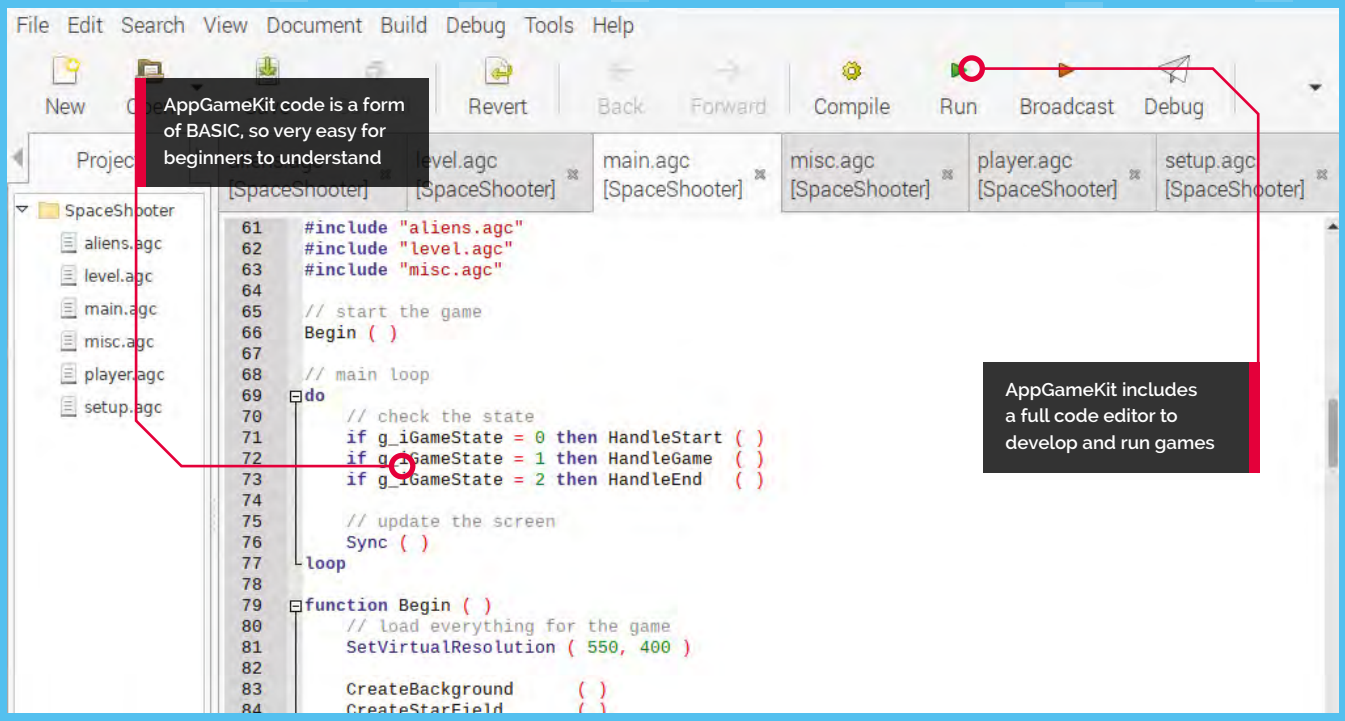
When you save your code, don't call it 'pygame' or Python will think that you are referring to the pygame module.

WHERE TO GET IDEAS

Did you know that *The MagPi's* sister magazine, Wireframe, features a section called Source Code every month with Pygame Zero game examples? **wfmag.cc**

MAKE GREAT GAMES WITH APPGAMEKIT

With AppGameKit you can develop professional-looking games, not just for Raspberry Pi but also for desktop and even mobile devices



Top Tip

GPIO pins

If you are feeling adventurous, you could try out the AGK features that allow you to read and write to and from the GPIO pins.

AppGameKit provides a cross-platform development system that was originally for PC desktops, but recently it has become available to download free for Raspberry Pi. You can use the same system on other platforms too, and develop on one system to run on a different one. You can even publish your games and earn money without paying any royalties. The engine has many tools to help you build your game, like 2D sprites, 3D, physics, sound, and even virtual reality. This guide will get you started with AppGameKit so that you can explore all the features.

Compatibility alert

Until recently, AppGameKit was compatible with all Raspberry Pi computers, but at the time of writing, it is difficult to get running on Raspberry Pi 4. Some system updates are needed for other Raspberry Pi computers, even with the latest version of Raspberry Pi OS. Make sure you have backed up any data from your Raspberry Pi microSD card before you start.

01 Get the download

First, we need to get the AppGameKit files. You'll need to go to the website **appgamekit.com** and sign up for an account. When that's done, go to the 'AppGameKit For Raspberry Pi' section in the 'Classic' menu item and download the editor files (they are free). Double-click the gzip file to open it and extract the files to somewhere suitable like your home directory. When it's unpacked, you will see a directory called **AGKPi**. Inside that, you will find the AGK launcher. Double-click to open the editor. If you want to see an error log when it's running, select 'Execute in Terminal' when prompted.

02 Doing the update

If you try to run any of the samples provided with AGK, you may find that you get some errors. This may only be an issue at the time of writing as there are regular updates available.

From a Terminal window, enter `sudo apt update`, followed by `sudo apt upgrade`, just to make sure we have everything up to date. Then, if you are getting errors about libgles2 (graphics library), type `sudo apt install libgles2-mesa libsdl2-dev`, which will install the necessary libraries. Then enter `sudo rpi-update` – this is a firmware update, so a bit more extreme than the usual updates, and the reason why you should make a backup of your memory card before issuing this command. Now reboot your Raspberry Pi.

03 Load a sample

A good sample to start with, to make sure everything is working, is the Space Shooter game. Select the Open icon on the toolbar of the editor and then navigate to the **SpaceShooter** directory, which is found in the **Games** folder inside **Projects**. Open the .agk file and you will see several files open in the editor. AGK uses a language very much like BASIC, so if you have used BASIC before you should be right at home. If you haven't learnt BASIC, then it's quite easy as it was designed for beginners.

04 Run the game

If everything has gone well with the install and updates, when you press the green Run arrow you should see a window open up titled AGK and a Start Game screen with spaceships floating about. If you don't see that, then check the Terminal window that launched the editor to see if there are any errors. You may see some warnings there anyway as some of the shader modes are not supported on Raspberry Pi, but the game should work fine. Start the game by clicking the screen and move the player ship up and down with cursor keys.

05 Make your program

Now you have the editor building a game, why not start your own? Start a new project by clicking the New icon on the toolbar. You will be asked for a name for the new project and a base path. Select the folder icon to the right of the base path input box, and navigate to somewhere suitable inside your home directory. Select Create and you will see a new file called **main.agc** open in the editor. In that file, there will already be a listing of the base code you need to start your game. Run it and you will see a black window open with the title 'test'.

With the AppGameKit samples, you can quickly see how to build many types of games

06 The sky's the limit

Have a look through the samples to see the range of what AGK can do. You will find a huge range of tutorials at **magpi.cc/agkyt** and there is a full user guide at **magpi.cc/agkguide**. There is also an active and helpful community forum at **magpi.cc/agkforum** where you will find more hints and tips to help you on your way. If you are having a problem with something, you'll find someone who has solved it and will tell you how. Don't be afraid to get stuck in and just start coding: the compiler will give you feedback on anything you get wrong.

Top Tip

Raspberry Pi 4

Feeling brave and want to run AGK on a Raspberry Pi 4? Check out the forum post at: **magpi.cc/agkpi4** for instructions.

You'll Need

- AppGameKit: **appgamekit.com**
- Raspberry Pi SDK: **appgamekit.com/agk-pi**

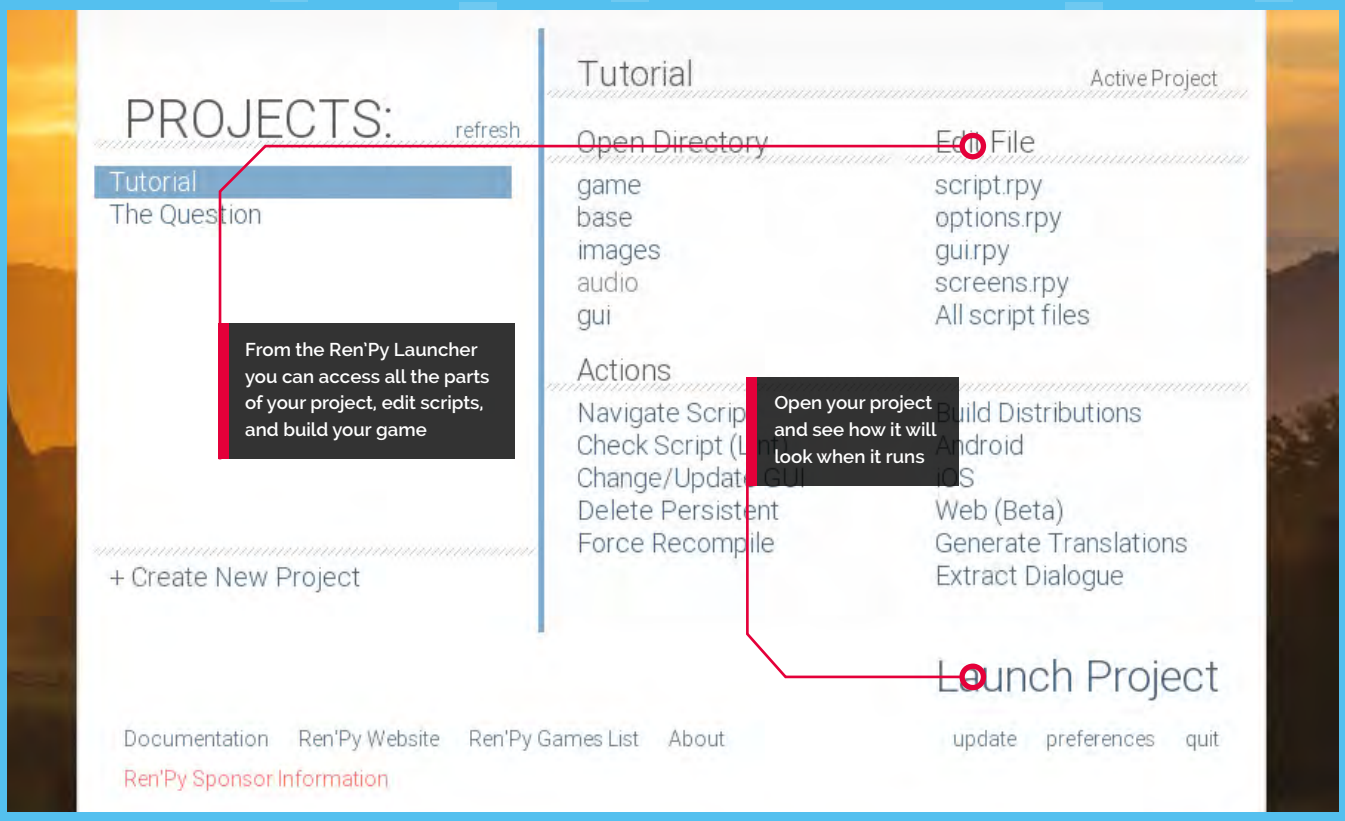
GRAPHICS RESOURCES

There are many graphics resources that are free to download. You can get images, animations and programs. Here are a few places to visit:

- **opengameart.org** has a wide range of artwork for backgrounds and character images to include in your games free of charge.
- **spriters-resource.com** specialises in sprites, which are the characters to include in games. They are often available in sprite sheets which have all the frames in one image file.
- **free3d.com** has many free (and paid-for) 3D models for you to download and use. There are models for just about any situation, some of them specifically designed for games.
- **gimp.org** is a great image manipulation program and should be all you need for creating 2D graphics for your games. It can be installed using `sudo apt install gimp` in a Terminal window.
- **blender.org** is best for creating a game with 3D graphics. Install Blender free from your Terminal with `sudo apt install blender`. Discover a range of Blender projects on Raspberry Pi's website (**magpi.cc/blenderprojects**).

MAKE ADVENTURE GAMES WITH REN'PY

This game engine is for storytelling. Use Raspberry Pi to combine words, images, and sounds to create interactive visual novels and life simulation games



Top Tip

Embed Python

Ren'Py scripting is quite similar to Python, but if you need to embed a Python program inside your Ren'Py game, you can do that too.

Ren'Py is open-source and free to download and use. You can even share your creations without paying a penny in royalties or licences. Ren'Py includes a simple scripting language to control the flow of your story and add interactivity to the pages. The engine also includes a wide selection of animation and transition effects to bring your games and graphic novels to life without needing to learn complicated animation software and supports the most common graphics and sound formats like JPG, PNG, MP3, and a whole lot more.

01 Get the files

First, download the install files from the Ren'Py website at **renpy.org/latest.html**. You will need the .bz2 version for Raspberry Pi. When it has downloaded, double-click to open the archive and extract it to a suitable place such as your home directory. You will also need to download and extract Raspberry Pi support files from the Additional Downloads section. Once this is all in place, you will find a file in the directory you have extracted called **renpy.sh**. Double-click this file and select 'Execute'. After a few seconds, you will see the Ren'Py Launcher open.

02 Tutorial time

Ren'Py includes a getting started tutorial, which is probably the best place to begin. By selecting the Tutorial project from the launcher, you will be introduced to Ren'Py's features by Eileen. She will show you how to start a new project and the ways to set colours and screen sizes. There are also sections in the tutorial to

▲ Ren'Py includes a tutorial where Eileen talks you through all the features of the system

cover adding your images, text, and sound to your pages. It then goes on to creating interactions and transitions to make your game engaging for your audience. Have a look at the Choices and Python section to see how scripting is used to ask questions and branch to different options.

03 Let's make a game

Going back to the Ren'Py Launcher, start a new project with the 'Create New Project' link on the left-hand side. You will then be asked where you want to save your project and what it should be named. Next, choose what screen resolution you want your production to use and the colour scheme that you would like. After a short pause for processing, your project will be created and listed with the tutorial in the Projects section in the Launcher.

04 Let's get scripting

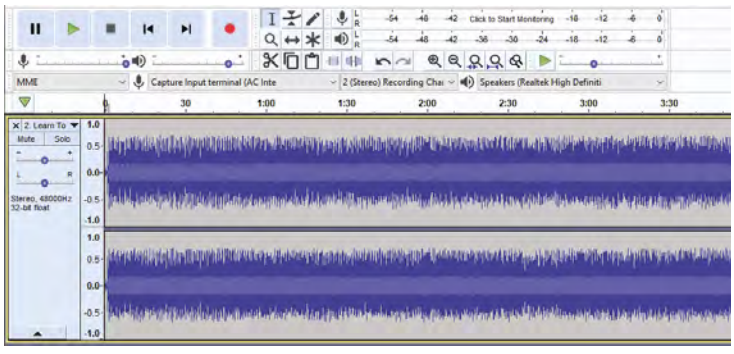
Start scripting the game by selecting the 'script.rpy' option under Edit File in the Launcher. It will ask you to select the editor you want to use and then open the script. From there you can make changes to the default script. When you want to test your changes, select your project and Launch Project, then select 'Start' from the list on the left of the window that opens. If you make changes to your script, you can then press **SHIFT+R** to reload your script and start the game again. If you need further help, select the Documentation link at the bottom left of the Launcher window, or check out the forums at **magpi.cc/renpyforum**.

SOUND RESOURCES

If you need to find sounds for your games, you can get a whole range of sound effects and background soundtracks from **freesound.org**, **zapsplat.com,** or **musopen.org/music**, all of which provide free downloads.

You may need to edit your sounds, in which case use Audacity – available for you to install using `sudo apt install audacity` from your Terminal window.

▼ Audacity enables you to edit sound files in a variety of formats such as WAV and MP3



Part 01

Get started with Pygame Zero

Pygame Zero is a great choice for anyone who wants to start writing computer games on Raspberry Pi

MAKER

Mark Vanstone

Educational software author from the nineties, author of the ArcVenture series, disappeared into the corporate software wasteland. Rescued by the Raspberry Pi!

magpi.cc/YiZnxl

@mindexplorers

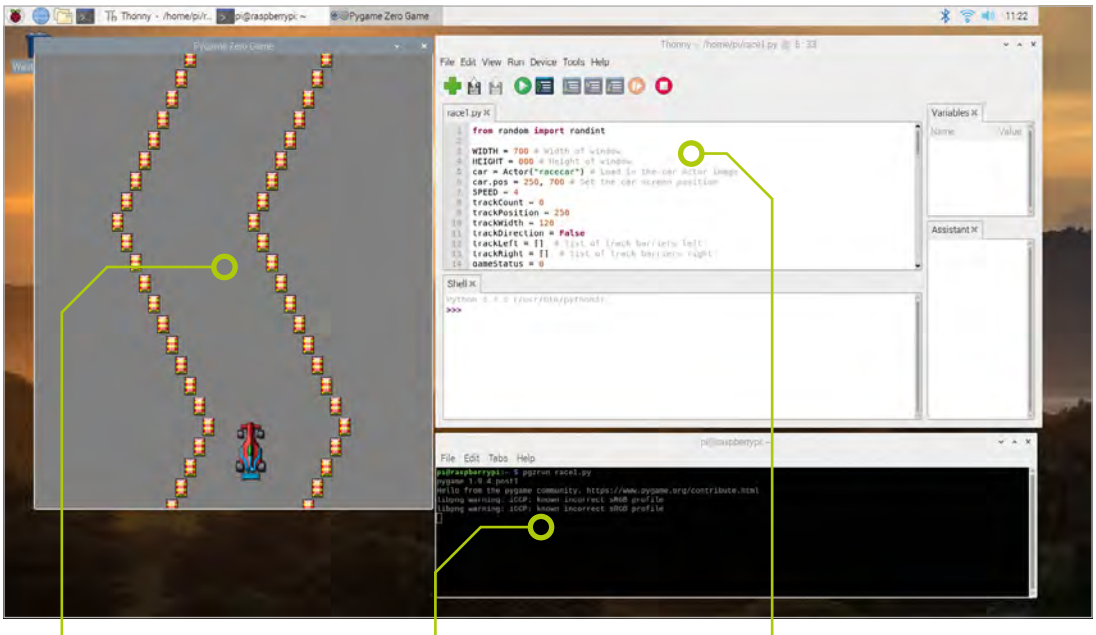
If you've done some Python coding and wanted to write a game, you may have come across Pygame. The Pygame module adds many functions that help you to write games in Python. Pygame Zero goes one step further to let you skip over the cumbersome process of making all those game loops and setting up your program structure. You don't need to worry about functions to load graphics or keeping data structures for all the game elements. If you just want to get stuck in and start making things happen on the screen without all the fluff, then Pygame Zero is what you need.

You'll Need

- An image manipulation program such as GIMP
- A little imagination
- A keyboard

01 Loading a suitable program editor

The first really labour-saving thing about Pygame Zero is that you can write a program in a simple text editor. We advise using the Thonny Python editor, as Pygame Zero needs to be formatted like Python with its indents and you'll get the benefit of syntax highlighting to help you along the way. So the first step in your journey will be to open Thonny, found in the Programming section of the Raspberry Pi OS main menu (click the raspberry icon). You'll be presented with a window featuring three panes.



02 Writing a Pygame Zero program

The top pane is where you will write your code. To start writing your first Pygame Zero program, click the Save icon and save your blank program – we suggest saving it as **pygame1.py** in your default user folder (just save the file without changing directory). And that's it: you have written your first Pygame Zero program! The Pygame Zero framework assumes that you will want to open a new window to run your game inside, so even a blank file will create a running game environment. Of course at this stage your game doesn't do very much, but you can test it to make sure that you can get a program running.

03 Running your first Pygame Zero program

With other Python programs, you can run them directly from the Python file window. While there is a method to enable you to do so with a Pygame Zero program (see part 2 of this tutorial series), let's use the simple alternative for now. All you need to do then is open a Terminal window from the main Raspbian menu, and type cd pygame-zero and type in `pgzrun pygame1.py` (assuming you called your program **pygame1.py**) and then hit **RETURN**. After a few seconds, a window titled 'Pygame Zero Game' should appear.

04 Setting up the basics

By default, the Pygame Zero window opens at the size of 800 pixels wide by 600 pixels high. If you want to change the size of your window, there are two predefined variables you can set. If you include `WIDTH = 700` in your program, then the window will be set at 700 pixels wide. If you include `HEIGHT = 800`, then the window will be set to 800 pixels high. In this tutorial we'll be writing a simple racing game, so we want our window to be a bit taller than it is wide. When you have set the `WIDTH` and `HEIGHT` variables, you could save your file as **race1.py** and test it like before by typing `pgzrun race1.py` into the Terminal window.

05 Look! No game loop!

When writing a Python game, normally you would have a game loop – that's a piece of code that is run over and over while the game is

race1.py

> Language: **Python**

DOWNLOAD THE FULL CODE:
magpi.cc/pgzero1

```
001.  from random import randint
002.  import pgzrun
003.
004.  WIDTH = 700 # Width of window
005.  HEIGHT = 800 # Height of window
006.  car = Actor("racecar") # Load in the car Actor image
007.  car.pos = 250, 700 # Set the car screen position
008.  SPEED = 4
009.  trackCount = 0
010.  trackPosition = 250
011.  trackWidth = 120
012.  trackDirection = False
013.  trackLeft = []  # list of track barriers left
014.  trackRight = []  # list of track barriers right
015.  gameStatus = 0
016.
017.  def draw(): # Pygame Zero draw function
018.      global gameStatus
019.      screen.fill((128, 128, 128))
020.      if gameStatus == 0:
021.          car.draw()
022.          b = 0
023.          while b < len(trackLeft):
024.              trackLeft[b].draw()
025.              trackRight[b].draw()
026.              b += 1
027.      if gameStatus == 1:
028.          # Red Flag
029.          screen.blit('rflag', (318, 268))
030.      if gameStatus == 2:
031.          # Chequered Flag
032.          screen.blit('cflag', (318, 268))
033.
034.  def update(): # Pygame Zero update function
035.      global gameStatus , trackCount
036.      if gameStatus == 0:
037.          if keyboard.left: car.x -= 2
038.          if keyboard.right: car.x += 2
039.          updateTrack()
040.      if trackCount > 200: gameStatus = 2 # Chequered flag state
041.
042.  def makeTrack(): # Function to make a new section of track
043.      global trackCount, trackLeft, trackRight, trackPosition, trackWidth
044.      trackLeft.append(Actor("barrier", pos = (trackPosition-trackWidth,0)))
045.      trackRight.append(Actor("barrier", pos = (trackPosition+trackWidth,0)))
046.      trackCount += 1
047.
048.  def updateTrack(): # Function to update where the track blocks appear
049.      global trackCount, trackPosition, trackDirection, trackWidth,
      gameStatus
050.      b = 0
051.      while b < len(trackLeft):
052.          if car.colliderect(trackLeft[b]) or car.colliderect(trackRight[b]):
053.              gameStatus = 1  # Red flag state
054.          trackLeft[b].y += SPEED
055.          trackRight[b].y += SPEED
056.          b += 1
057.      if trackLeft[len(trackLeft)-1].y > 32:
058.          if trackDirection == False: trackPosition += 16
059.          if trackDirection == True: trackPosition -= 16
060.          if randint(0, 4) == 1: trackDirection = not trackDirection
061.          if trackPosition > 700-trackWidth: trackDirection = True
062.          if trackPosition < trackWidth: trackDirection = False
063.          makeTrack()
064.
065.  # End of functions
066.  makeTrack() # Make first block of track
067.
068.  pgzrun.go()
```

To respond to key presses, Pygame Zero has a built-in object called keyboard. The arrow key states can be read with keyboard.up, keyboard.down, and so on

figure1.py

Language: **Python**

```
001.  WIDTH = 700
002.  HEIGHT = 800
003.
004.  def draw():
005.      screen.fill((128, 128, 128))
```

Figure 1 To set the height and width of a Pygame Zero window, just set the variables `HEIGHT` and `WIDTH`. Then you can fill the screen with a colour

running. Pygame Zero does away with this idea and provides predefined functions to handle each of the tasks that the game loop normally performs. The first of these we will look at is the function `draw()`. We can write this function into our program the same as we would normally define a function in Python, which is `def draw():`. Then, so that you can see the `draw` function doing something, add a line underneath indented by one tab: `screen.fill((128, 128, 128))`. This is shown in the **figure1.py** listing overleaf.

Top Tip

The graphics

If you use PNG files for your graphics rather than JPGs, you can keep part of the image transparent.

06 The Python format

You may have noticed that in the previous step we said to indent the `screen.fill` line by one tab. Pygame Zero follows the same formatting rules as Python, so you will need to take care to indent your code correctly. The indents in Python show that the code is inside a structure. So if you define a function, all the code inside it will be indented by one tab. If you then have a condition or a loop, for example an `if` statement, then the contents of that condition will be indented by another tab (so two in total).

07 All the world's a stage

The screen object used in Step 5 is a predefined object that refers to the window we've opened for our game. The `fill` function fills the window with the RGB value (a tuple value) provided – in this case, a shade of grey. Now that we have our stage set, we can create our actors. Actors in Pygame Zero are dynamic graphic objects, much the same as sprites in other programming systems. We can load an actor by typing `car = Actor("racecar")`. This is best placed near the top of your program, before the `draw()` function.

08 It's all about image

When we define an actor in our program, what we are actually doing is saying 'go and get this image'. In Pygame Zero our images need to be stored in a directory called **images**, next to our program file. So our actor would be looking for an image file in the **images** folder called **racecar.png**. It could be a GIF or a JPG file, but it is recommended that your images are PNG files as that file type provides good-quality images with transparencies. You can get a full free image creation program called GIMP by typing `sudo apt-get install gimp` in your Terminal window. If you want to use our images, you can download them from **magpi.cc/pgzero1**.

09 Drawing your Actor

Once you have loaded in your image by defining your actor, you can set its position on the screen. You can do this straight after loading the actor by typing `car.pos = 250, 500` to set it at position 250, 500 on the screen. Now, when the `draw()` function runs, we want to display our race car at the coordinates that we have set. So, in our `draw()` function, after the `screen.fill` command we can type `car.draw()`. This will draw our race car at its defined position. Test your program to make sure this is working, by saving it and running `pgzrun race1.py`, as before.

10 I'm a control freak!

Once we have our car drawing on the screen, the next stage is to enable the player to move it backwards and forwards. We can do this with key presses; in this case we are going to use the left and right arrow keys. We can read the state of these keys inside another predefined function called `update()`. We can type in the definition of this function by adding `def update():` to our program. This function is continually checked while the game is running. We can now add an indented `if` statement to check the state of a key; e.g., `if keyboard.left:`.

11 Steering the car

We need to write some code to detect key presses of both arrow keys and also to do something if we detect that either has been pressed. Continuing from our `if keyboard.left:` line, we can write `car.x -= 2`. This means subtract 2 from the car's x coordinate. It could also be written in long-hand as `car.x = car.x - 2`. Then, on the next line and with the same indent as the first `if` statement, we can do the same for the right arrow; i.e., `if keyboard.right: car.x += 2`. These lines of code will move the car actor left and right.

12 The long and winding road

Now that we have a car that we can steer, we need a track for it to drive on. We are going to build our track out of actors, one row at a time. We will need to make some lists to keep track of the actors we create. To create our lists, we can write the following near the top of our program: `trackLeft = []` (note the square brackets) and then, on the next line, `trackRight = []`. This creates two empty lists: one to hold the data about the left side of the track, and one to hold the data about the right-hand side.

figure2.py

Language: **Python**

```
001. def makeTrack(): # Function to make a new section of track
002.     global trackCount, trackLeft, trackRight,
     trackPosition, trackWidth
003.     trackLeft.append(Actor("barrier", pos =
     (trackPosition-trackWidth,0)))
004.     trackRight.append(Actor("barrier", pos =
     (trackPosition+trackWidth,0)))
005.     trackCount += 1
```

▲ **Figure 2** The makeTrack() function. This creates two new Actors with the barrier image at the top of the screen

13 Building the track

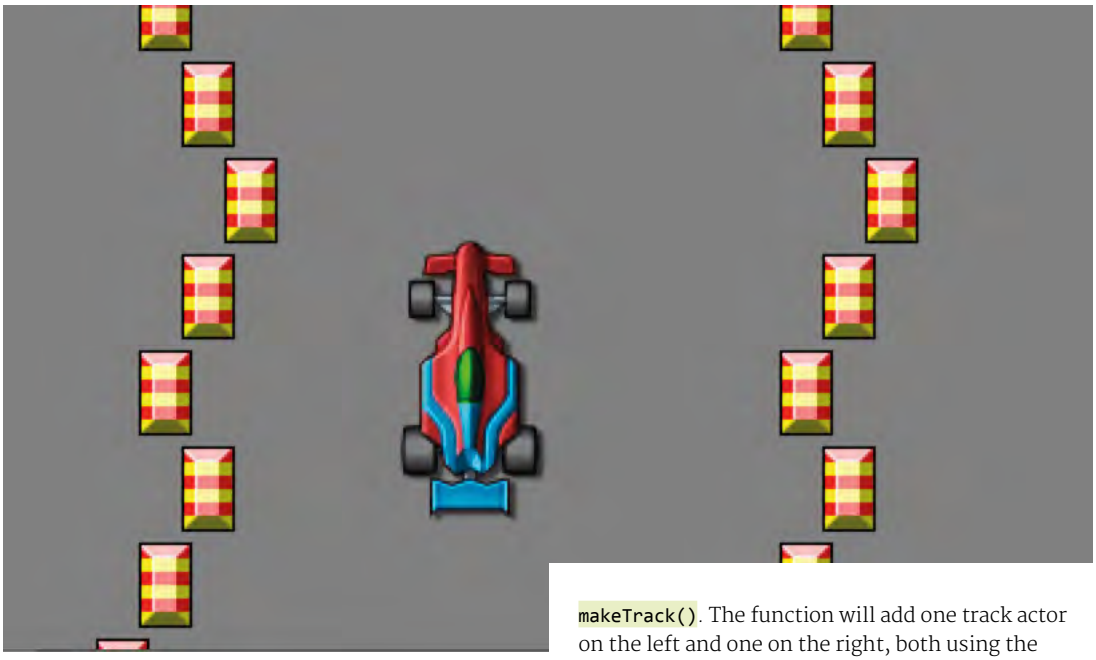
We will need to set up a few more variables for the track. After your two lists, declare the following variables: `trackCount = 0` and then `trackPosition = 250`, then `trackWidth = 120`, and finally `trackDirection = false`. Then let's make a new function called `makeTrack()`. Define this function after your `update()` function. See the **figure2.py** listing for the code to put inside

figure3.py

Language: **Python**

```
001. def updateTrack(): # Function to update where the track
     blocks appear
002.     global trackCount, trackPosition, trackDirection,
     trackWidth
003.     b = 0
004.     while b < len(trackLeft):
005.         trackLeft[b].y += SPEED
006.         trackRight[b].y += SPEED
007.         b += 1
008.     if trackLeft[len(trackLeft)-1].y > 32:
009.         if trackDirection == False: trackPosition += 16
010.         if trackDirection == True: trackPosition -= 16
011.         if randint(0, 4) == 1: trackDirection = not
     trackDirection
012.         if trackPosition > 700-trackWidth: trackDirection =
     True
013.         if trackPosition < trackWidth: trackDirection = False
014.         makeTrack()
```

▼ **Figure 3** The updateTrack() function. Notice the constant SPEED – we need to define this at the top of our program, perhaps starting with the value 4



The race car with barriers making up a track to stay within. The track pieces are created by random numbers so each play is different

Figure 4 The `draw()` function and the `update()` function with conditions (`if` statements) to do different things depending on the value of `gameStatus`

figure4.py

Language: **Python**

```
001.  def draw(): # Pygame Zero draw function
002.      global gameStatus
003.      screen.fill((128, 128, 128))
004.      if gameStatus == 0:
005.          car.draw()
006.          b = 0
007.          while b < len(trackLeft):
008.              trackLeft[b].draw()
009.              trackRight[b].draw()
010.              b += 1
011.      if gameStatus == 1:
012.          # Red Flag
013.
014.      if gameStatus == 2:
015.          # Chequered Flag
016.
017.  def update(): # Pygame Zero update function
018.      global gameStatus , trackCount
019.      if gameStatus == 0:
020.          if keyboard.left: car.x -= 2
021.          if keyboard.right: car.x += 2
022.          updateTrack()
023.      if trackCount > 200: gameStatus = 2 # Chequered
      flag state
```

`makeTrack()`. The function will add one track actor on the left and one on the right, both using the image **barrier.png**. Each time we call this function, it will add a section of track at the top of the screen.

14 On the move

The next thing that we need to do is to move the sections of track down the screen towards the car. Let's write a new function called `updateTrack()`. We will call this function in our `update()` function after we do the keyboard checks. See the **figure3.py** listing for the code for our `updateTrack()` function. In this function we are using `randint()`. This is a function that we must load from an external module, so at the top of our code we write `from random import randint`. We use this function to make the track curve backwards and forwards.

15 Making more track

Notice at the bottom of the `updateTrack()` function, there is a call to our `makeTrack()` function. This means that for each update when the track sections move down, a new track section is created at the top of the screen. We will need to start this process off, so we will put a call to `makeTrack()` at the bottom of our code. If we run our code at the moment, we should see a track snaking down towards the car. The only problem is that we can move the car over the track barriers

and we want to keep the car inside them with some collision detection.

16 A bit of a car crash

We need to make sure that our car doesn't touch the track actors. As we are looking through the existing barrier actors in our `updateTrack()` function, we may as well test for collisions at the same time. We can write `if car.colliderect(trackLeft[b]) or car.colliderect(trackRight[b]):` and then, indented on the next line, `gameStatus = 1`. We have not covered `gameStatus` yet – we'll use this variable to show if the game is running, the car has crashed, or we've reached the end of the race. Define your `gameStatus` variable near the top of the program as `gameStatus = 0`. You will also need to add it to the global variables in the `updateTrack()` function.

17 Changing state

In this game we will have three different states to the game stored in our variable `gameStatus`. The first or default state will be that the game is running and will be represented by the number 0. The next state will be set if the car crashes, which will be the number 1. The third state will be if we have finished the race, which we'll set as the number 2 in `gameStatus`. We will need to reorganise our `draw()` function and our `update()` function to respond to the `gameStatus` variable. See the **figure4.py** listing for how we do that.

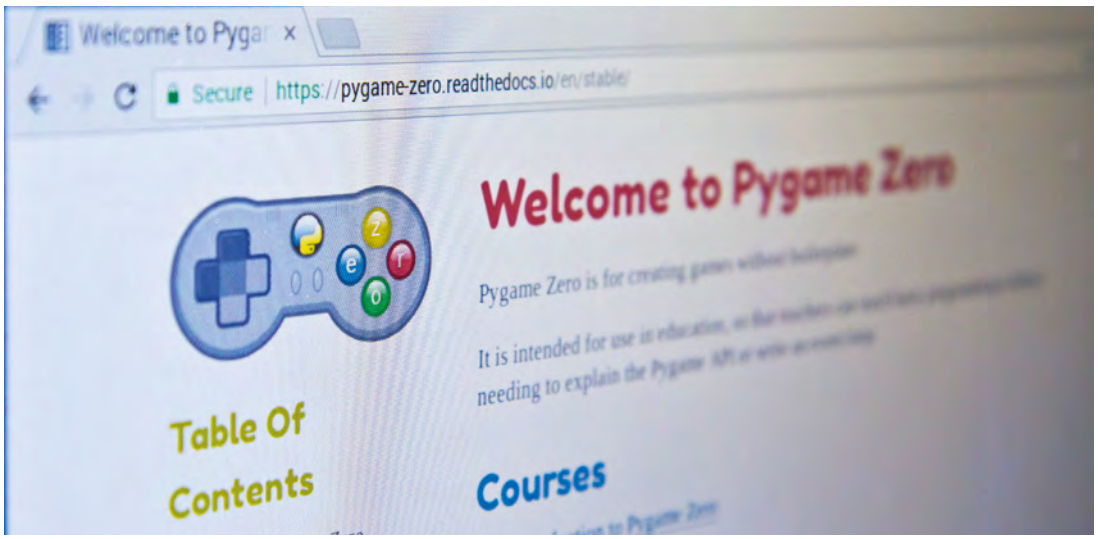
Each of the barrier blocks is checked against the car to detect collisions. If the car hits a barrier, the red flag graphic is displayed

18 Finishing touches

All we need to do now is to display something if `gameStatus` is set to 1 or 2. If `gameStatus` is 1 then it means that the car has crashed and we should display a red flag. We can do that with the code: `screen.blit('rflag', (318, 268))`. To see if the car has reached the finish, we should count how many track sections have been created and then perhaps when we get to 200, set `gameStatus` to 2. We can do this in the `update()` function as in **figure4.py**. Then, in the `draw()` function, if the `gameStatus` is 2, then we can write `screen.blit('cflag', (318, 268))`. Have a look at the full code listing to see how this all fits together.

19 Did you win?

If you didn't get the program working first time, you are not alone – it's quite rare to have everything exactly right first time. Check that you have written all the variables and functions correctly and that the capital letters are in the right places. Python also insists on having code properly formatted with indents. When it's all in place, test your program as before and you should have a racing game with a chequered flag at the end!



The official Pygame Zero documentation can be found at **magpi.cc/fBqznh**

Top Tip

Run from IDE

Since the upgrade to version 1.2, programs can be run straight from Thonny by adding `import pgzrun` to the top of the code and `pgzrun.go()` at the bottom.

Top Tip

Changing the speed

If you want to make the track move faster or slower, try changing the value of `SPEED` at the start of the program.

Part 02

Pygame Zero
Simple Brian

Recreate a classic electronic game using Pygame Zero

You'll Need

- An image manipulation program such as GIMP, or images from **magpi.cc/pgzero2**
- The latest version of Pygame Zero
- A good memory

Long, long ago, before Raspberry Pi existed, there was a game. It was a round plastic box with four coloured buttons on top and you had to copy what it did. To reconstruct this classic game using Pygame Zero, we'll first need a name. To avoid any possible trademark misunderstandings and because we are using the Python language, let's call it 'Brian'. The way the game works is that Brian will show you a colour sequence by lighting up the buttons and then you have to copy the sequence by pressing the coloured buttons in the same sequence. Each round, an extra button is added to the sequence and you get a point for each round you complete correctly. The game continues until you get the sequence wrong.

01 Run, run as fast as you can

In the previous tutorial (page 48), we ran code by typing the `pgzrun` command in a Terminal window. With the 1.2 update of Pygame Zero, however, there is now a way to run your programs directly from a Python editor such as Thonny, by adding a couple of lines of code (see **figure1.py**).

02 The stage is set

We'll need some images that make up the buttons of the Brian game. You can make your own or get ours from GitHub at **magpi.cc/pgzero2**. The images will need to be in an **images** directory next to your program file. We have called our starting images **redunlit**, **greenunlit**, **blueunlit**, and **yellowunlit** because all the buttons will be unlit at the start of the game. We have also got a play button so that the player can click on it to start the game.

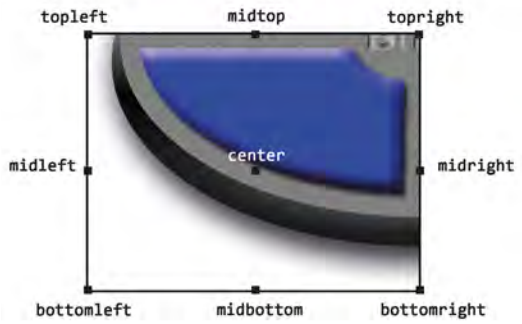
03 Getting the actors on stage

We can create actors by supplying an image name and a position on the screen for it to go. There are several ways of supplying the position information. This time we'll use position handles to define where the character appears. We will

figure1.py

Language: **Python**

```
001. import pgzrun
002.
003. # Your program code will go here
004.
005. pgzrun.go()
```



▲ There are nine positions that an Actor's co-ordinates can be aligned to when the Actor is created

use the same coordinates for each quadrant of the whole graphic, but we'll change the handle names we use. For these actors we can use `bottomright`, `bottomleft`, `topright`, and `topleft`, as well as the coordinates (400,270), which is the centre point of our whole graphic. Take a look at **figure2.py**.

04 Look at the state of that

We now need to add some logic to determine if each button is on or off and show it lit or unlit accordingly. We can do this by adding a variable to each of our actors. We want it to be either on or off, so we can set this variable as a Boolean value, i.e. True or False. If we call this variable `state`, we can add it to the Actor by writing (for the first button): `myButtons[0].state = False`. We then do the same for each of the button actors with their list numbers 1, 2, and 3 because we defined them as a list.

05 Light goes on, light goes off

We have defined a state for each button; now we have to write some code to react to that state. First, let's make a couple of lists which hold the names of the images we will use for the two states. The first list will be the images we use for the buttons being lit, which would be: `buttonsLit = ['redlit', 'greenlit', 'bluelit', 'yellowlit']`. We then need a list of the unlit buttons: `buttonsUnlit = ['redunlit', 'greenunlit', 'blueunlit', 'yellowunlit']`. Then we can use these lists in an `update()` function to set the image of each button to match its state. See **figure3.py**.

06 Switching images

We can see from **figure3.py** that each time our `update()` function runs, we will loop through

figure2.py

DOWNLOAD THE FULL CODE: magpi.cc/pgzero2

> Language: **Python**

```
001. import pgzrun
002.
003. myButtons = []
004. myButtons.append(Actor('redunlit', bottomright=(400,270)))
005. myButtons.append(Actor('greenunlit', bottomleft=(400,270)))
006. myButtons.append(Actor('blueunlit',topright=(400,270)))
007. myButtons.append(Actor('yellowunlit',topleft=(400,270)))
008. playButton = Actor('play', pos=(400,540))
009.
010. def draw(): # Pygame Zero draw function
011.     screen.fill((30, 10, 30))
012.     for b in myButtons: b.draw()
013.     playButton.draw()
014.
015. pgzrun.go()
```

figure3.py

> Language: **Python**

```
001. import pgzrun
002.
003. myButtons = []
004. myButtons.append(Actor('redunlit', bottomright=(400,270)))
005. myButtons[0].state = False
006. myButtons.append(Actor('greenunlit', bottomleft=(400,270)))
007. myButtons[1].state = False
008. myButtons.append(Actor('blueunlit',topright=(400,270)))
009. myButtons[2].state = False
010. myButtons.append(Actor('yellowunlit',topleft=(400,270)))
011. myButtons[3].state = False
012. buttonsLit = ['redlit', 'greenlit', 'bluelit', 'yellowlit']
013. buttonsUnlit = ['redunlit', 'greenunlit', 'blueunlit',
     'yellowunlit']
014. playButton = Actor('play', pos=(400,540))
015.
016. def draw(): # Pygame Zero draw function
017.     screen.fill((30, 10, 30))
018.     for b in myButtons: b.draw()
019.     playButton.draw()
020.
021. def update(): # Pygame Zero update function
022.     bcount = 0
023.     for b in myButtons:
024.         if b.state == True: b.image = buttonsLit[bcount]
025.         else: b.image = buttonsUnlit[bcount]
026.         bcount += 1
027.
028. pgzrun.go()
```

figure4.py

Language: **Python**

```
001.  def on_mouse_down(pos):
002.      global myButtons
003.      for b in myButtons:
004.          if b.collidepoint(pos): b.state = True
005.
006.  def on_mouse_up(pos):
007.      global myButtons
008.      for b in myButtons: b.state = False
```

figure5.py

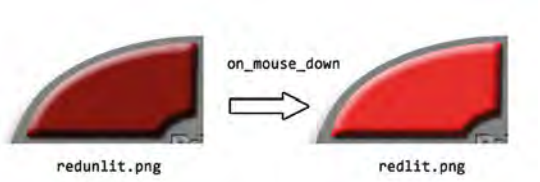
Language: **Python**

```
001.  def playAnimation():
002.      global playPosition, playingAnimation
003.      playPosition = 0
004.      playingAnimation = True
005.
006.  def addButton():
007.      global buttonList
008.      buttonList.append(randint(0, 3))
009.      playAnimation()
```

figure6.py

Language: **Python**

```
001.  def update(): # Pygame Zero update function
002.      global myButtons, playingAnimation, playPosition
003.      if playingAnimation:
004.          playPosition += 1
005.          listpos = math.floor(playPosition/LOOPDELAY)
006.          if listpos == len(buttonList):
007.              playingAnimation = False
008.              clearButtons()
009.          else:
010.              litButton = buttonList[listpos]
011.              if playPosition%LOOPDELAY > LOOPDELAY/2:
      litButton = -1
012.              bcount = 0
013.              for b in myButtons:
014.                  if litButton == bcount: b.state = True
015.                  else: b.state = False
016.                  bcount += 1
```



▲ When the mouse is clicked on a button, we switch the unlit image for the lit image

our button list. If the button's state is `True`, we set the image of the button to the image in the `buttonsLit` list. If not (i.e. the state variable is `False`), we set the image of the button to the image in the `buttonsUnlit` list.

07 What happens if I press this button?

We need to write a way to allow the user to press the buttons and make them light up. We can do this with the Pygame Zero functions `on_mouse_down()` and `on_mouse_up()`. If the mouse has been clicked down, we should set our button state to `True`. We also need to test if the mouse button has been released; if so, all the buttons should be set to `False`. We can test the value we are passed (`pos`) into these functions with the method `collidepoint()`, which is part of the actor object.

08 Ups and downs

We can write a test in `on_mouse_down()` for each button, to see if it has been pressed, and then change the state of the button if it has been pressed. We can then write code to set all the button states to `False` in the `on_mouse_up()` function, and our `update()` and `draw()` functions will now reflect what we need to see on the screen from those actions. Look at **figure4.py** and you will see how we can change the state of the buttons as a response to mouse events. When you have added this to your program, test it to make sure that the buttons light up correctly when clicked.

09 Write a list

Now that we have our buttons working, we will need to make a way to use them in two different ways. The first will be for the game to display a sequence for the player to follow, and the second is to receive input from the player when they repeat the sequence. For the first task we will need to build a list to represent the sequence and then play that sequence to the player. Let's define our list at the top of the code with `buttonList = []` and then make a function `def addButton():` which will create an additional entry into the sequence each round.

10 That's a bit random

We can generate our sequence by generating random integers using the `random` module. We can use this module by importing it at the top of our code with: `from random import randint`. As we only need the `randint()` function, we import that function specifically. To add a new number to the sequence in the `addButton()` function, we can write `buttonList.append(randint(0, 3))`, which will add a number between 0 and 3 to our list. Once we have added our new number, we will want to show the player the sequence, so add a line in the `addButton()` function: `playAnimation()`.

11 Playing the animation

We have set up our function to create the sequence. Now we need a system to play the latter so the player can see it. We'll do this with a counter variable called `playPosition`. We define this at the start of our code: `playPosition = 0`. We'll also need a variable to show that our animation is playing: `playingAnimation = False`, also written at the start of the code. We can then define our `playAnimation()` function that we used in the previous step. Look at **figure5.py** to see how the `addButton()` and `playAnimation()` functions are written.

12 Are we playing?

So, once we have set our animation going, we will need to react to that in our `update()` function. We know that all we need to do is change the state of the buttons and the `draw()` function will handle the visuals for us. In our `update()` function, we have to say: "If the animation is playing then increment our animation counter, check that we haven't reached the end of the animation and if not then light the button (change its state to True) which is indicated by our sequence list." This is a bit of a mouthful, so in **figure6.py** we can see how this translates into code.

13 Getting a bit loopy

We can see from **figure6.py** that we are incrementing the play position each time `update()` is called. What we want to do is keep each button in the sequence lit for several refreshes, so we divide the `playPosition` by a predefined number (`LOOPDELAY`) to get our list position that we want to display. We round the result downwards with the `math.floor()` function (to use this, we import the `math` module at the top of the code). So if `LOOPDELAY` is 80, we'll move from one list position (`listpos`) to the next every 80 times `update()` is called.

14 A dramatic pause

Still in **figure6.py**, we check to see if we have reached the end of the `buttonList` with `listpos`. If we have then we stop the animation. Otherwise, if we are still running the animation, we work out which button should be lit from our `buttonList`. We could just say "light that button and set the rest to unlit", but before we do that we have a line that basically says: "If we are in the second half of our button lighting loop, set all the buttons to unlit." This means that we will get a pause in between each button being lit when no buttons are lit. We can then just loop through our buttons and set their state according to the value of `litButton`.

15 Testing the animation

Now, ignoring the fact that we have a play button ready and waiting to do something, we can test our animation by calling the `addButton()` function. This function adds a random button number to the list and sets the animation in motion. To test it, we can call it a few times at the bottom of our code, just above `pgzrun.go()`. If we call the `addButton()` function three times then three numbers will be added to the `buttonList` and the animation will start. If this all works, we are ready to add the code to capture the player's response.

16 I need input

We can collect the player's clicks on the buttons just by adding another list, `playerInput`, to the definitions at the top of the code and adding a few lines into our `on_mouse_down()` function. Add a counter variable `bcount = 0` at the top of the function and then add one to `bcount` at the end of the loop. Then, after `if b.collidepoint(pos):` we add `playerInput.append(bcount)`. We can then test the player input to see if it matches the `buttonList` we are looking for. We will write this as a separate function called `checkPlayerInput()` and call it at the end of our `on_mouse_down()` function. As we now have the basis of our game, refer to the

Top Tip

Globals

You can read global variables inside a function, but if you change the value of the variable, you must declare it as global in the function.

Top Tip

Modulo or %

The % symbol is used to get the remainder after a division calculation. It's useful for creating smaller repeats within a larger loop.

Top Tip

Changing the delay

We have used the constant LOOPDELAY for timing our loops, if the game is running too slow, decrease this value at the top of the code.

full listing to see how the rest of the code comes together as we go through the final steps.

17 Game over man

The `checkPlayerInput()` function will check the buttons that the player has clicked against the list held in `buttonList`, which we have been building up with the `addButton()` function. So we need to loop through the `playerInput` list with a counter variable – let's call it `ui`, and write `if playerInput[ui] != buttonList[ui]: gameOver()`. If we get to the end of the list and both `playerInput` and `buttonList` are the same length then we know that the player has completed the sequence and we can signal that the score needs to be incremented. The `score` variable is defined at the top of the code as `score = 0`. In our `on_mouse_up()` function, we can then respond to the score signal by incrementing the score and setting the next round in motion.

18 Just press play

We still haven't done anything with that play button actor that we set up at the beginning. Let's put some code behind that to get the game started. Make sure you have removed any testing calls at the bottom of your code to `addButton()` (Step 15). We'll need a variable to check if the game is started, so put `gameStarted = False` at the top of the code with the other variables and then in our `on_mouse_up()` function we can add a test: `if playButton.collidepoint(pos) and gameStarted == False:` and then set the `gameStarted` variable to `True`. We can set a countdown variable when the play button is clicked so that there is a slight pause before the first animation starts.

Top Tip

Using text

If you are going to use text, it's a good idea to display some on the first screen as it can take a while to load fonts for the first time. You may get unwanted delays if you load it later.

19 Finishing touches

We're nearly there with our game: we have a way to play a random sequence and build that list round by round, and we have a way to capture and check user input. The last things we need are some instructions for the player, which we can do with the Pygame Zero `screen.draw.text()` function. We will want an initial 'Press Play to Start' message, a 'Watch' message for when the animation is playing, a 'Now You' message to prompt the player to respond, and a score message to be displayed when the game is over. Have a look in the `draw()` function in the complete listing to see how these fit in.

There are many ways we can enhance our game; for example, the original electronic game featured sound – something we cover later (see page 74).

brian.py

Language: **Python 3**

```
001. import pgzrun
002. from random import randint
003. import math
004. WIDTH = 800
005. HEIGHT = 600
006.
007. myButtons = []
008. myButtons.append(Actor('redunlit',
     bottomright=(400,270)))
009. myButtons[0].state = False
010. myButtons.append(Actor('greenunlit',
     bottomleft=(400,270)))
011. myButtons[1].state = False
012. myButtons.append(Actor('blueunlit',
     topright=(400,270)))
013. myButtons[2].state = False
014. myButtons.append(Actor('yellowunlit',
     topleft=(400,270)))
015. myButtons[3].state = False
016. buttonsLit = ['redlit', 'greenlit',
     'bluelit', 'yellowlit']
017. buttonsUnlit = ['redunlit',
     'greenunlit', 'blueunlit',
     'yellowunlit']
018. playButton = Actor('play',
     pos=(400,540))
019. buttonList = []
020. playPosition = 0
021. playingAnimation = False
022. gameCountdown = -1
023. LOOPDELAY = 80
024. score = 0
025. playerInput = []
026. signalScore = False
027. gameStarted = False
028.
029. def draw(): # Pygame Zero draw
     function
030.     global playingAnimation, score
031.     screen.fill((30, 10, 30))
032.     for b in myButtons: b.draw()
033.     if gameStarted:
034.         screen.draw.text("Score
     : " + str(score), (310, 540),
     owidth=0.5, ocolor=(255,255,255),
     color=(255,128,0) , fontsize=60)
035.     else:
036.         playButton.draw()
037.         screen.draw.
     text("Play", (370, 525),
     owidth=0.5, ocolor=(255,255,255),
     color=(255,128,0) , fontsize=40)
038.         if score > 0:
039.             screen.draw.text("Final
```

```
    Score : " + str(score), (250, 20), owidth=0.5,
    ocolor=(255,255,255), color=(255,128,0) ,
    fontsize=60)
040.         else:
041.             screen.draw.text("Press Play to Start",
    (220, 20), owidth=0.5, ocolor=(255,255,255),
    color=(255,128,0) , fontsize=60)
042.     if playingAnimation or gameCountdown > 0:
043.         screen.draw.text("Watch", (330, 20),
    owidth=0.5, ocolor=(255,255,255), color=(255,128,0)
    , fontsize=60)
044.     if not playingAnimation and gameCountdown == 0:
045.         screen.draw.text("Now You", (310, 20),
    owidth=0.5, ocolor=(255,255,255), color=(255,128,0)
    , fontsize=60)
046.
047. def update(): # Pygame Zero update function
048.     global myButtons, playingAnimation,
    playPosition, gameCountdown
049.     if playingAnimation:
050.         playPosition += 1
051.         listpos = math.floor(playPosition/LOOPDELAY)
052.         if listpos == len(buttonList):
053.             playingAnimation = False
054.             clearButtons()
055.         else:
056.             litButton = buttonList[listpos]
057.             if playPosition%LOOPDELAY > LOOPDELAY/2:
    litButton = -1
058.             bcount = 0
059.             for b in myButtons:
060.                 if litButton == bcount: b.state = True
061.                 else: b.state = False
062.                 bcount += 1
063.     bcount = 0
064.     for b in myButtons:
065.         if b.state == True: b.image =
    buttonsLit[bcount]
066.         else: b.image = buttonsUnlit[bcount]
067.         bcount += 1
068.     if gameCountdown > 0:
069.         gameCountdown -=1
070.         if gameCountdown == 0:
071.             addButton()
072.             playerInput.clear()
073.
074. def gameOver():
075.     global gameStarted, gameCountdown, playerInput,
    buttonList
076.     gameStarted = False
077.     gameCountdown = -1
078.     playerInput.clear()
079.     buttonList.clear()
080.     clearButtons()
081.
082. def checkPlayerInput():
083.     global playerInput, gameStarted, score,
    buttonList, gameCountdown, signalScore
084.     ui = 0
085.     while ui < len(playerInput):
086.         if playerInput[ui] != buttonList[ui]:
    gameOver()
087.         ui += 1
088.     if ui == len(buttonList): signalScore = True
089.
090. def on_mouse_down(pos):
091.     global myButtons, playingAnimation,
    gameCountdown, playerInput
092.     if not playingAnimation and gameCountdown == 0:
093.         bcount = 0
094.         for b in myButtons:
095.             if b.collidepoint(pos):
096.                 playerInput.append(bcount)
097.                 b.state = True
098.             bcount += 1
099.         checkPlayerInput()
100.
101. def on_mouse_up(pos):
102.     global myButtons, gameStarted, gameCountdown,
    signalScore, score
103.     if not playingAnimation and gameCountdown == 0:
104.         for b in myButtons: b.state = False
105.     if playButton.collidepoint(pos) and gameStarted
    == False:
106.         gameStarted = True
107.         score = 0
108.         gameCountdown = LOOPDELAY
109.     if signalScore:
110.         score += 1
111.         gameCountdown = LOOPDELAY
112.         clearButtons()
113.         signalScore = False
114.
115. def clearButtons():
116.     global myButtons
117.     for b in myButtons: b.state = False
118.
119. def playAnimation():
120.     global playPosition, playingAnimation
121.     playPosition = 0
122.     playingAnimation = True
123.
124. def addButton():
125.     global buttonList
126.     buttonList.append(randint(0, 3))
127.     playAnimation()
128.
129. pgzrun.go()
```

Part 03

Pygame Zero
PiVaders

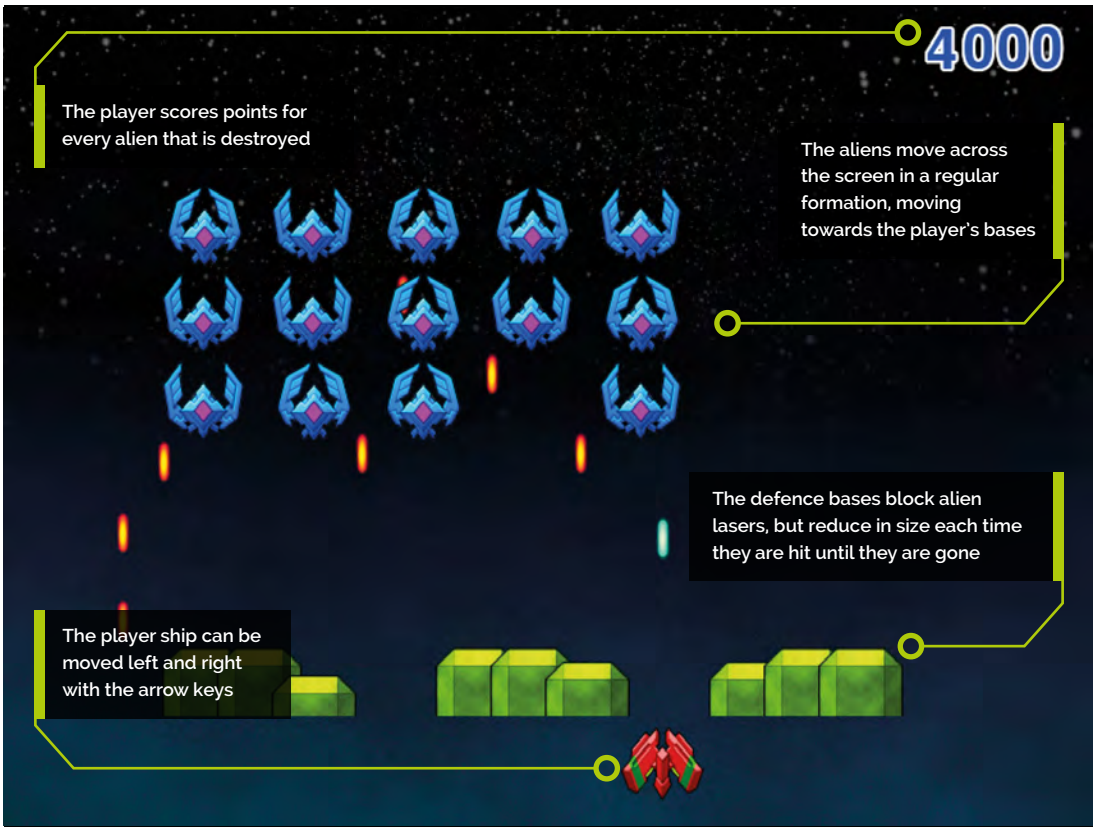
There must be very few people who have not played a shooting game, and for some it may have been their very first experience of a computer game

You'll Need

- An image manipulation program such as GIMP, or images from **magpi.cc/pgzero4**
- The latest version of Pygame Zero
- A cool head as the lasers rain down on you

The shooting-style game format requires quite a few different coding techniques to make it work. For some time, if your author needed to learn a new coding language, he would task himself to write an invaders game in it. This would give a good workout through the syntax and functions of the language.

This tutorial will be split into two parts. In the first we will build a basic shooting game with aliens, lasers, defence bases, and a score. The second part (page 98) will add all the extra bits that make it into a game similar to the one that appeared in amusement arcades and sports halls in the late 1970s.



01 Let's get stuck in

If you have read the previous tutorials, you will know how we set up a basic Pygame Zero program, so we can jump right in to getting things on the screen. We will need some graphics for the various elements of the game – you can design them yourself or use ours from: **magpi.cc/pgzero4**. The Pygame Zero default screen size is 800 width by 600 height, which is a good size for this game, so we don't need to define `WIDTH` or `HEIGHT`.

02 A bit of a player

Let's start with getting the player ship on the screen. If we call our graphic **player.png**, then we can create the player Actor near the top of our code by writing `player = Actor("player", (400, 550))`.

We will probably want something a bit more interesting than just a plain black window, so we can add a background in our `draw()` function. If we draw this first, everything else that we draw will be on top of it. We can draw it using the `blit()` function by writing `screen.blit('background', (0, 0))` – assuming we have called our background image **background.png**. Then, to draw the player, just add `player.draw()` afterwards.

03 Let's get moving

We need the player ship to respond to key presses, so we'll check the Pygame Zero keyboard object to see if certain keys are currently pressed. Let's make a new function to deal with these inputs. We will call the function `checkKeys()` and we'll need to call it from our `update()` function.

In the `checkKeys()` function, we write `if keyboard.left:` and then `if player.x > 40: player.x -= 5`. We need to declare the player Actor object as global inside our `checkKeys()` function. We then write a similar piece of code to deal with the right arrow key; **figure1.py** shows how this all fits together.

04 An alien concept

We now want to create a load of aliens in formation. You can have them in whatever format you want, but we'll set up three rows of aliens with six on each row. We have an image called **alien.png** and can make an Actor for each alien that we will store in a list so that we can easily loop through the list to perform actions on them. When we create the alien Actors, we will use a bit of maths to set the initial x and y coordinates. It would be a good idea to define a function to set up the aliens – `initAliens()` – and because we will want to set up other elements too, we could define a function `init()`, from which we can call all the setup functions.

05 Doing the maths

To position our aliens and to create them as Actors, we can declare a list – `aliens = []` – and then create a loop using `for a in range(18):`. In this loop, we need to create each Actor and then work out where their x and y coordinates will be to start. We can do this in the loop by writing: `aliens.append(Actor("alien1", (210+(a % 6)*80,100+(int(a/6)*64))))`. This may look a little daunting, but we can break it down by saying 'x is 210 plus the remainder of dividing by 6 multiplied by 80'.

This will provide us with x coordinates starting at 210 and with a spacing of 80 between each. The y calculation is similar, but we use normal division, make it an integer, and multiply by 64.

figure1.py

```
001. import pgzrun
002.
003. player = Actor("player", (400, 550)) # Load in the player
     Actor image
004.
005. def draw(): # Pygame Zero draw function
006.     screen.blit('background', (0, 0))
007.     player.draw()
008.
009. def update(): # Pygame Zero update function
010.     checkKeys()
011.
012. def checkKeys():
013.     global player
014.     if keyboard.left:
015.         if player.x > 40: player.x -= 5
016.     if keyboard.right:
017.         if player.x < 760: player.x += 5
018.
019. pgzrun.go()
```

▲ Functions to create a player ship and background, display them, and handle moving the player ship

figure2.py

```
001.  def updateAliens():
002.      global moveSequence, moveDelay
003.      movex = movey = 0
004.      if moveSequence < 10 or moveSequence > 30: movex = -15
005.      if moveSequence == 10 or moveSequence == 30:
006.          movey = 50
007.      if moveSequence >10 and moveSequence < 30: movex = 15
008.      for a in range(len(aliens)):
009.          animate(aliens[a], pos=(aliens[a].x + movex,
      aliens[a].y + movey), duration=0.5, tween='linear')
010.          if randint(0, 1) == 0:
011.              aliens[a].image = "alien1"
012.          else:
013.              aliens[a].image = "alien1b"
014.      moveSequence +=1
015.      if moveSequence == 40: moveSequence = 0
```

▲ The `updateAliens()` function. Calculate the movement for the aliens based on the variable `moveSequence`

06 Believing the strangest things

After that slightly obscure title reference, we shall introduce the idea of the alien having a status. As we have seen in previous instalments, we can add extra data to our Actors, and in this case we will want to add a status variable to the alien after we have created it. We'll explain how we are going to use this a bit later. Now it's time to get the little guys on the screen and ready for action. We can write a simple function called `drawAlien()` and just loop through the alien list to draw them by writing: `for a in range(len(aliens)): aliens[a].draw()` . Call the `drawAlien()` function inside the `draw()` function.

07 The aliens are coming!

We are going to create a function that we call inside our `update()` function that keeps track of what should happen to the aliens. We'll call it `updateAliens()`. We don't want to move the aliens every time the update cycle runs, so we'll keep a counter called moveCounter and increment it each `update()`; then, if it gets to a certain value (`moveDelay`), we will zero the counter. If the counter is zero, we call `updateAliens()`. The `updateAliens()` function will calculate how much they need to move in the x and y directions to get them to go backwards and forwards across the screen and move down when they reach the edges.

08 Updating the aliens

To work out where the aliens should move, we'll make a counter loop from 0 to 40. From 0 to 9 we'll move the aliens left, on 10 we'll move them down, then from 11 to 29 move them right. On 30 they move down and then from 31 to 40 move left. Have a look at **figure2.py** to see how we can do this in the `updateAliens()` function and how that function fits into our `update()` function. Notice how we can use the Pygame Zero function `animate()` to get them to move smoothly. We can also add a switch between images to make their legs move.

09 All your base are belong to us

Now we are going to build our defence bases. There are a few problems to overcome in that we want to construct our bases from Actors, but there are no methods for clipping an Actor when it is displayed. Clipping is a term to describe that we only display a part of the image. This is a method we need if we are going to make the bases shrink as they are hit by alien lasers. What we will have to do is add a function to the Actor, just like we have added extra variables to them before.

10 Build base

We will make three bases which will be made of three Actors each. If we wanted to display the whole image (**base1.png**), we would create a list of base Actors and display each Actor with some code like `bases[0].draw()`. What we want to do is add a variable to the base to show how high we want it to be. We will also need to write a new function to draw the base according to the height variable. Have a look at **figure3.py** to see how we write the new function and attach it to each Actor. This means we can now call this function from each base Actor using: `bases[b].drawClipped()`, as shown in the `drawBases()` function.

11 Can I shoot something now?

To make this into a shooting game, let's add some lasers. We need to fire lasers from the player ship and also from the aliens, but we are going to keep them all in the same list. When we create a new laser by making an Actor and adding it to the

Top Tip

Beware of deleting elements of a list

If you delete a list element while you are looping through it with `range (len(list))`, when you get to the end of the loop it will run out of elements and return an error because the range of the loop is the original length of the list.

list `lasers[]`, we can give the Actor a type. In this case we'll make alien lasers type 0 and player lasers type 1. We'll also need to add a status variable. The creation and updating of the lasers is similar to other elements we've looked at; **figure4.py** (overleaf) shows the functions that we can use.

12 Making the lasers work

You can see in **figure4.py** that we can create a laser from the player by adding a check for the **SPACE** key being pressed in our `checkKeys()` function. We will use the blue laser image called **laser2.png**. Once the new laser is in our list of lasers, it will be drawn to the screen if we call the `drawLasers()` function inside our `draw()` function. In our `updateLasers()` function we loop through the list of lasers and check which type it is. So if it is type 1 (player), we move the laser up the screen and then check to see if it hit anything. Notice the calls to a `listCleanup()` function at the bottom. We will come to this in a bit.

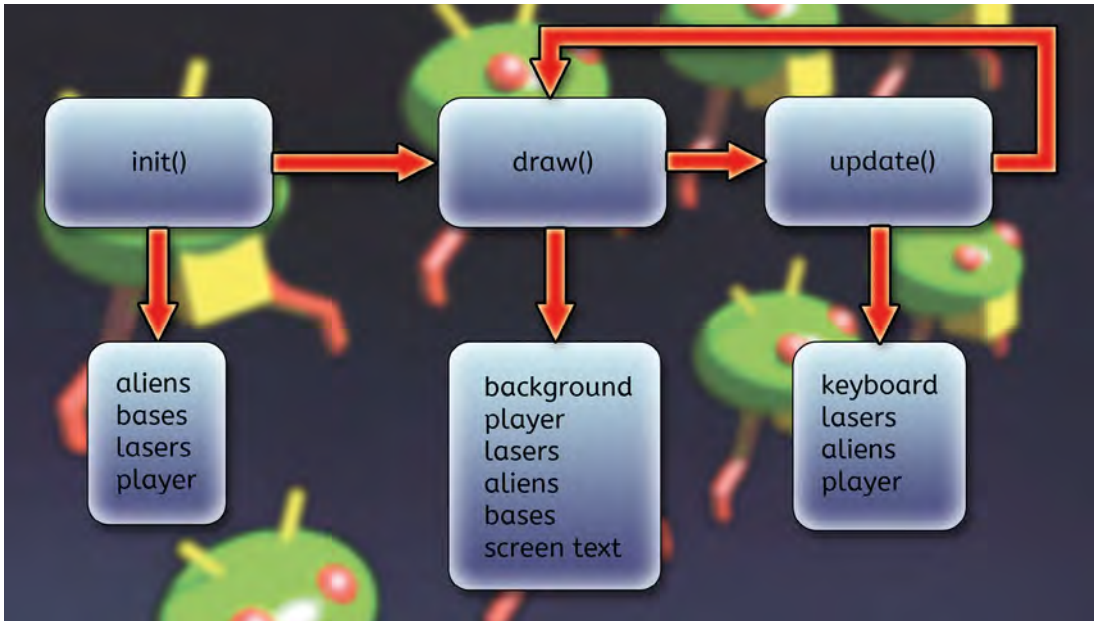
13 Collision course

Let's look at `checkPlayerLaserHit()` first. We can detect if the laser has hit any aliens by looping round the alien list and checking with the Actor function – `collidepoint((lasers[l].x, lasers[l].y))` – to see if a collision has occurred. If an alien has been hit, this is where our status variables come into play. Rather than just removing the laser and the alien from their lists, we need to flag them as ready to remove. The reason for this is that if we remove anything from a list while we are

figure3.py

```
001.  def drawClipped(self):
002.      screen.surface.blit(self._surf, (self.x-32, self.y-
      self.height+30),(0,0,64,self.height))
003.
004.  def initBases():
005.      global bases
006.      bases = []
007.      bc = 0
008.      for b in range(3):
009.          for p in range(3):
010.              bases.append(Actor("base1",
      midbottom=(150+(b*200)+(p*40),520)))
011.              bases[bc].drawClipped = drawClipped.__get__
      (bases[bc])
012.              bases[bc].height = 60
013.              bc +=1
014.
015.  def drawBases():
016.      for b in range(len(bases)): bases[b].drawClipped()
```

▲ Setting up an extension function to draw an Actor with clipping



Top Tip

Write functions for each collective action

To make coding easier to read rather than having lots of code associated with one type of element in the `draw()` or `update()` functions, send it out to a function like `drawLasers()` or `checkKeys()`.

figure4.py

```
001.  def checkKeys():
002.      global player, lasers
003.      if keyboard.space:
004.          l = len(lasers)
005.          lasers.append(Actor("laser2",
      (player.x,player.y-32)))
006.          lasers[l].status = 0
007.          lasers[l].type = 1
008.
009.  def drawLasers():
010.      for l in range(len(lasers)): lasers[l].draw()
011.
012.  def updateLasers():
013.      global lasers, aliens
014.      for l in range(len(lasers)):
015.          if lasers[l].type == 0:
016.              lasers[l].y += (2*DIFFICULTY)
017.              checkLaserHit(l)
018.              if lasers[l].y > 600: lasers[l].status = 1
019.          if lasers[l].type == 1:
020.              lasers[l].y -= 5
021.              checkPlayerLaserHit(l)
022.              if lasers[l].y < 10: lasers[l].status = 1
023.      lasers = listCleanup(lasers)
024.      aliens = listCleanup(aliens)
```

▲ Checking the keys that are pressed, creating lasers, moving them, and checking if they have collided with anything

looping through any of the lists then by the time we get to the end of the list, we are an element short and an error will be created. So we set these Actors to be removed with `status` and then remove them afterwards with `listCleanup()`.

Top Tip

Collect all your setup code in one place

If possible, it is good to have as much of the code that sets everything back to the beginning in one place so that you can easily restart the game.

14 Cleaning up the mess

The `listCleanup()` function creates a new empty list, then runs through the list that is passed to it, only transferring items to the new list that have a status of 0. This new list is then returned back and used as the list going forward. Now that we have made a system for one type of laser we can easily adapt that for our alien laser type. We can create the alien lasers in the same way as the player lasers, but instead of waiting for a keyboard press we can just produce them at random intervals using `if randint(0, 5) == 0:` when we are updating our aliens. We set the type to 0 rather than 1 and move them down the screen in our `updateLasers()` function.

15 Covering the bases

So far, we haven't looked at what happens when a laser hits one of the defence bases. Because we are changing the height of the base Actors, the built-in collision detection won't give us the result we want, so we need to write another custom function to check laser collision on the base Actor. Our new function, `collideLaser()` will check the laser coordinates against the base's coordinates, taking into account the height of the base. We then attach the new function to our base Actor when it is created. We can use the new `collideLaser()` function for checking both the player and the alien lasers and remove the laser if it hits – and if it is an alien laser, reduce the height of the base that was hit.

16 Laser overkill

We may want to change the number of lasers being fired by the aliens, but at the moment our player ship gets to fire a laser every `update()` cycle. If the **SPACE** key is held down, a constant stream of lasers will be fired, which not only is a little bit unfair on the poor aliens but will also take its toll on the speed of the game. So we need to put some limits on the firing speed and we can do this with another built-in Pygame Zero object: the clock. If we add a variable `laserActive` to our player Actor and set it to zero when it fires, we can then call `clock.schedule(makeLaserActive, 1.0)` to call the function `makeLaserActive()` after 1 second.

17 I'm hit! I'm hit!

We need to look now at what happens when the player ship is hit by a laser. For this we will make a multi-frame animation. We have five explosion images to put into a list, with our normal ship image at the beginning, and attach it to our player Actor. We need to import the Math module, then in each `draw()` cycle we write: `player.image = player.images[math.floor(player.status/6)]`, which will display the normal ship image while `player.status` is 0. If we set it to 1 when the player ship is hit, we can start the animation in motion. In the `update()` function we write: `if player.status > 0: player.status += 1`. As the status value increases, it will start to draw the sequence of frames one after the other.

18 Initialisation

Now, it may seem a bit strange to be dealing with initialisation near the end of the tutorial, but we have been adding and changing the structure of our game elements as we have gone along and only now can we really see all the data that we need to set up before the game starts. In Step 04 we created a function called `init()` that we should call to get the game started. We could also use this function to reset everything back to start the game again. If we have included all the initialisation functions and variables we have talked about, we should have something like **figure5.py**.

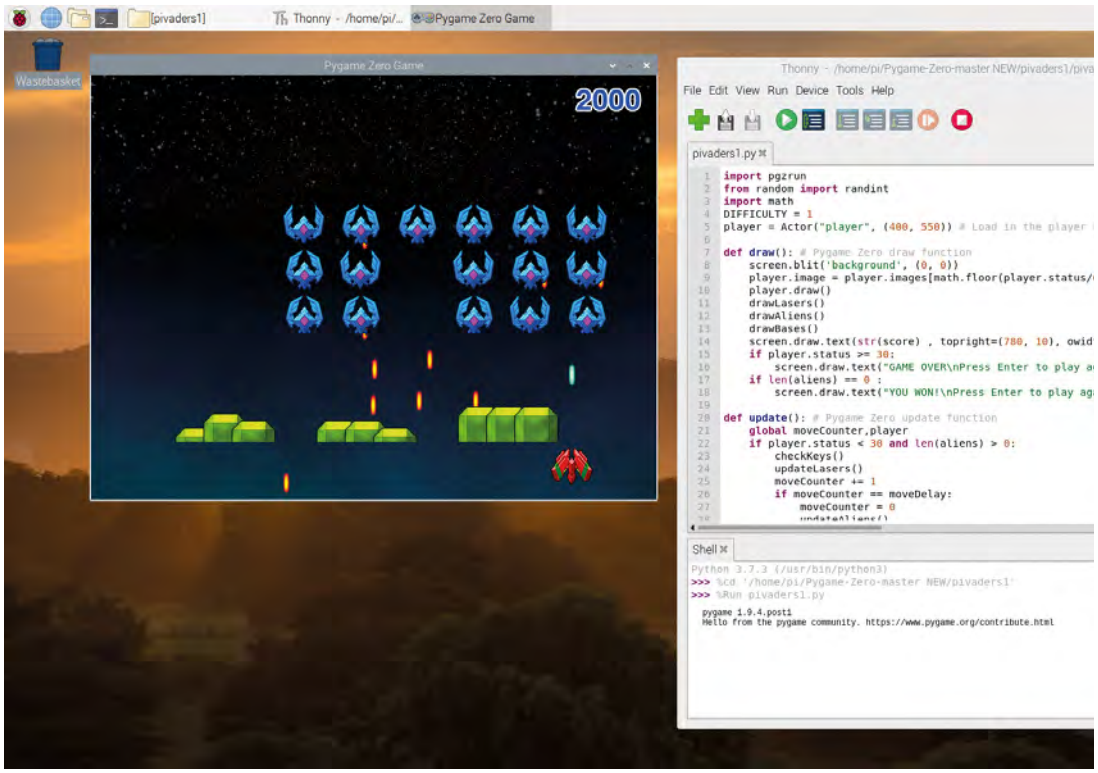
figure5.py

```
001. def init():
002.     global lasers, score, player, moveSequence,
     moveCounter, moveDelay
003.     initAliens()
004.     initBases()
005.     moveCounter = moveSequence = player.status = score =
     player.laserCountdown = 0
006.     lasers = []
007.     moveDelay = 30
008.     player.images = ["player","explosion1","explosion2",
     "explosion3","explosion4","explosion5"]
009.     player.laserActive = 1
```

▲ The initialisation of our data. Calling this function sets our variables back to their start values

19 They're coming in too fast!

There are a few finishing touches to do to complete this first part. We can set a `DIFFICULTY` value near the top of the code and use it on various elements to make the game harder. We should also add a score, which we do by adding 1000 to a global variable score if an alien is hit, and then display that in the top right of the screen in the `draw()` function. When the game finishes (the player has been hit or all the aliens are gone), we should display a suitable message. Have a look at the complete listing to see how these bits fit in. When that's all done, we should have the basis of a Space Invaders game. In the next part, we will add more into the game, such as levels, lives, sound, bonus aliens, and a leaderboard.



◀ It's game over for now, but we'll be back in the second part to improve the game

Top Tip

Define several variables at once

If you are setting several variables to the same value, you can combine them into one line by writing `a = b = c = 0` to set a, b, and c to zero.

pivaders1.py

```
import pgzrun
from random import randint
import math
DIFFICULTY = 1
player = Actor("player", (400, 550)) # Load in the player Actor image

def draw(): # Pygame Zero draw function
    screen.blit('background', (0, 0))
    player.image = player.images[math.floor(player.status/6)]
    player.draw()
    drawLasers()
    drawAliens()
    drawBases()
    screen.draw.text(str(score), topright=(780, 10), owidth=0.5, ocolor=(255,255,255), color=(0,64,255), fontsize=60)
    if player.status >= 30:
        screen.draw.text("GAME OVER\nPress Enter to play again" , center=(400, 300), owidth=0.5, ocolor=(255,255,255), color=(255,64,0), fontsize=60)
    if len(aliens) == 0 :
        screen.draw.text("YOU WON!\nPress Enter to play again" , center=(400, 300), owidth=0.5, ocolor=(255,255,255), color=(255,64,0) , fontsize=60)

def update(): # Pygame Zero update function
    global moveCounter,player
    if player.status < 30 and len(aliens) > 0:
        checkKeys()
        updateLasers()
        moveCounter += 1
        if moveCounter == moveDelay:
            moveCounter = 0
            updateAliens()
        if player.status > 0: player.status += 1
    else:
        if keyboard.RETURN: init()

def drawAliens():
    for a in range(len(aliens)): aliens[a].draw()

def drawBases():
    for b in range(len(bases)):
        bases[b].drawClipped()

def drawLasers():
    for l in range(len(lasers)): lasers[l].draw()

def checkKeys():
    global player, lasers
    if keyboard.left:
        if player.x > 40: player.x -= 5
    if keyboard.right:
        if player.x < 760: player.x += 5
    if keyboard.space:
        if player.laserActive == 1:
            player.laserActive = 0
            clock.schedule(makeLaserActive, 1.0)
            l = len(lasers)
            lasers.append(Actor("laser2", (player.x,player.y-32)))
            lasers[l].status = 0
            lasers[l].type = 1

def makeLaserActive():
    global player
    player.laserActive = 1

def checkBases():
    for b in range(len(bases)):
        if l < len(bases):
            if bases[b].height < 5:
                del bases[b]

def updateLasers():
    global lasers, aliens
    for l in range(len(lasers)):
        if lasers[l].type == 0:
            lasers[l].y += (2*DIFFICULTY)
            checkLaserHit(l)
            if lasers[l].y > 600:
                lasers[l].status = 1
        if lasers[l].type == 1:
            lasers[l].y -= 5
            checkPlayerLaserHit(l)
            if lasers[l].y < 10:
                lasers[l].status = 1
    lasers = listCleanup(lasers)
    aliens = listCleanup(aliens)

def listCleanup(l):
    newList = []
    for i in range(len(l)):
        if l[i].status == 0: newList.append(l[i])
    return newList

def checkLaserHit(l):
    global player
```

DOWNLOAD THE FULL CODE:
magpi.cc/pgzero4

```
092.      if player.collidepoint((lasers[l].x,
      lasers[l].y)):
093.          player.status = 1
094.          lasers[l].status = 1
095.      for b in range(len(bases)):
096.          if bases[b].collideLaser(lasers[l]):
097.              bases[b].height -= 10
098.              lasers[l].status = 1
099.
100.  def checkPlayerLaserHit(l):
101.      global score
102.      for b in range(len(bases)):
103.          if bases[b].collideLaser(lasers[l]):
104.              lasers[l].status = 1
105.      for a in range(len(aliens)):
106.          if aliens[a].collidepoint((lasers[l].x,
      lasers[l].y)):
107.              lasers[l].status = 1
108.              aliens[a].status = 1
109.              score += 1000
110.
111.  def updateAliens():
112.      global moveSequence, lasers, moveDelay
113.      movex = movey = 0
114.      if moveSequence < 10 or moveSequence > 30:
115.          movex = -15
116.      if moveSequence == 10 or moveSequence == 30:
117.          movey = 50 + (10 * DIFFICULTY)
118.          moveDelay -= 1
119.      if moveSequence >10 and moveSequence < 30:
120.          movex = 15
121.      for a in range(len(aliens)):
122.          animate(aliens[a], pos=(aliens[a].x + movex,
      aliens[a].y + movey), duration=0.5, tween='linear')
123.          if randint(0, 1) == 0:
124.              aliens[a].image = "alien1"
125.          else:
126.              aliens[a].image = "alien1b"
127.              if randint(0, 5) == 0:
128.                  lasers.append(Actor("laser1",
      (aliens[a].x,aliens[a].y)))
129.                  lasers[len(lasers)-1].status = 0
130.                  lasers[len(lasers)-1].type = 0
131.          if aliens[a].y > 500 and player.status ==
      0:
132.              player.status = 1
133.      moveSequence +=1
134.      if moveSequence == 40: moveSequence = 0
135.
136.  def init():
137.      global lasers, score, player, moveSequence,
      moveCounter, moveDelay
138.      initAliens()
139.      initBases()
140.      moveCounter = moveSequence = player.status =
      score = player.laserCountdown = 0
141.      lasers = []
142.      moveDelay = 30
143.      player.images =
      ["player","explosion1","explosion2",
      "explosion3","explosion4","explosion5"]
144.      player.laserActive = 1
145.
146.  def initAliens():
147.      global aliens
148.      aliens = []
149.      for a in range(18):
150.          aliens.append(Actor("alien1", (210+
151.  (a % 6)*80,100+(int(a/6)*64))))
152.          aliens[a].status = 0
153.
154.  def drawClipped(self):
155.      screen.surface.blit(self._surf, (self.x-32,
      self.y-self.height+30),(0,0,64,self.height))
156.
157.  def collideLaser(self, other):
158.      return (
159.          self.x-20 < other.x+5 and
160.          self.y-self.height+30 < other.y and
161.          self.x+32 > other.x+5 and
162.          self.y-self.height+30 + self.height >
      other.y
163.      )
164.
165.  def initBases():
166.      global bases
167.      bases = []
168.      bc = 0
169.      for b in range(3):
170.          for p in range(3):
171.              bases.append(Actor("base1",
      midbottom=(150+(b*200)+(p*40),520)))
172.              bases[bc].drawClipped =
      drawClipped.__get__(bases[bc])
173.              bases[bc].collideLaser =
      collideLaser.__get__(bases[bc])
174.              bases[bc].height = 60
175.              bc +=1
176.
177.  init()
178.  pgzrun.go()
```

Part 04

Pygame Zero
PiVaders: part 2

This arcade shooter may be the first computer game that springs to mind for a lot of people. Here in part two we will take our basic PiVaders game from part one and add all the extras

You'll Need

- An image manipulation program such as GIMP, or images from **magpi.cc/pgzero5**
- The latest version of Pygame Zero
- The Audacity sound editor or similar or sounds available from **magpi.cc/pgzero5**
- Speakers or headphones

In part one, last issue, we set up the basics for our PiVaders single-screen shoot-'em-up with our player ship controlled by the keyboard, defence bases, the aliens moving backwards and forwards across the screen, and lasers flying everywhere. In this part we will add lives and levels to the game, introduce a bonus alien, code a leader board for high scores, and add some groovy sound effects. We may even get round to adding an introduction screen if we get time. We are going to start from where we left off in part one. If you don't have the part one code and files, you can download them from GitHub at **magpi.cc/pgzero4**.

01 You only live thrice

It was a tradition with Space Invaders to be given three lives at the start of the game. We can easily set up a place to keep track of our player lives by writing `player.lives = 3` in our `init()` function. While we are in the `init()` function, let's add a player name variable with `player.name = ""` so that we can show names on our leader board, but we'll come to that in a bit. To display the number of lives our player has, we can add `drawLives()` to our `draw()` function and then define our `drawLives()` function containing a loop which 'blits' **life.png** once for each life in the top left of the screen.

02 Life after death

Now we have a counter for how many lives the player has, we will need to write some code to deal with what happens when a life is lost. In part one we ended the game when the `player.status` reached 30. In our `update()` function we already have a condition to check the `player.status` and if there are any aliens still alive. Where we have written `if player.status == 30:` we can write `player.lives -=1`. We can also check to see if the player has run out of lives when we check to see if the **RETURN** (aka **ENTER**) key is pressed.

03 Keep calm and carry on

Once we have reduced `player.lives` by one and the player has pressed the **RETURN** key, all we need to do to set things back in motion is to set `player.status = 0`. We may want to reset the laser list too, because if the player was hit by a flurry of lasers we may find that several lives are lost without giving the player a chance to get out of the way of subsequent lasers. We can do this by writing `lasers = []`. If the player has run out of lives at this point, we will send them off to the leader-board page. See **figure1.py** to examine the code for dealing with lives.

04 On the level

The idea of having levels is to start the game in an easy mode; then, when the player has shot all the aliens, we make a new level which is a bit harder than the last. In this case we are going to tweak a few variables to make each level more difficult. To start, we can set up a global variable `level = 1` in our `init()` function. Now we can use our `level` variable to alter things as we increase the value. Let's start by speeding up how quickly the aliens move down the screen as the level goes up. When we calculate the `movey` value in `updateAliens()`, we can write `movey = 40 + (5*level)` on the condition that `moveSequence` is 10 or 30.

05 On the up

To go from one level to the next, the player will need to shoot all the aliens. We can tell if there are any aliens left if `len(aliens) = 0`. So, with that in mind, we can put a condition in our `draw()` function with if `len(aliens) == 0:` and then draw text on the screen to say that the level has been cleared. We can put the same condition in the section of the `update()` function where we are waiting for **RETURN** to be pressed. When **RETURN** is pressed and the length of the aliens list is 0, we can add 1 to `level` and call `initAliens()` and `initBases()` to set things ready to start the new level.

figure1.py

```
001.  def draw()
002.      # additional drawing code
003.      drawLives()
004.      if player.status >= 30:
005.          if player.lives > 0:
006.              drawCentreText(
      "YOU WERE HIT!\nPress Enter to re-spawn")
007.          else:
008.              drawCentreText(
      "GAME OVER!\nPress Enter to continue")
009.
010.  def init():
011.      # additional player variables
012.      player.lives = 3
013.      player.name = ""
014.
015.  def drawLives():
016.      for l in range(player.lives):
017.          screen.blit("life", (10+(l*32),10))
018.
019.  def update():
020.      # additional code for life handling
021.      global player, lasers
022.      if player.status < 30 and len(aliens) > 0:
023.          if player.status > 0:
024.              player.status += 1
025.              if player.status == 30:
026.                  player.lives -= 1
027.      else:
028.          if keyboard.RETURN:
029.              if player.lives > 0:
030.                  player.status = 0
031.                  lasers = []
032.              else:
033.                  # go to the leader-board
034.                  pass;
035.
036.  def drawCentreText(t):
037.      screen.draw.text(t , center=(400, 300), owidth=0.5,
      ocolor=(255,255,255), color=(255,64,0) , fontsize=60)
```

▲ Code to deal with player lives. Notice the `drawCentreText()` function to short-cut printing text to the centre of the screen

figure2.py

```
001.  def updateBoss():
002.      global boss, level, player, lasers
003.      if boss.active:
004.          boss.y += (0.3*level)
005.          if boss.direction == 0: boss.x -= (1* level)
006.          else: boss.x += (1* level)
007.          if boss.x < 100: boss.direction = 1
008.          if boss.x > 700: boss.direction = 0
009.          if boss.y > 500:
010.              sounds.explosion.play()
011.              player.status = 1
012.              boss.active = False
013.          if randint(0, 30) == 0:
014.              lasers.append(Actor("laser1",
      (boss.x,boss.y)))
015.              lasers[len(lasers)-1].status = 0
016.              lasers[len(lasers)-1].type = 0
017.      else:
018.          if randint(0, 800) == 0:
019.              boss.active = True
020.              boss.x = 800
021.              boss.y = 100
022.              boss.direction = 0
```

▲ Code to update the boss or bonus alien. This code runs when the boss is active or uses random numbers to see if it's time to make it active

06 Front and centre

You may have noticed in **figure1.py** that we made a couple of calls to a function called `drawCentreText()` which we have not yet discussed. All that this function does is to shorten the process of writing text to the centre of the screen. We assume that the text will be positioned at coordinates (400, 300) and will have a set of standard style settings and colours, and the function definition just contains one line: `screen.draw.text(t , center=(400, 300), owidth=0.5, ocolor=(255,255,255), color=(255,64,0), fontsize=60)` – where `t` is passed into the function as a parameter.

07 Flying like a boss

To liven up our game a little bit, we are going to add in a bonus or boss alien. This could be triggered in various ways, but in this case we will start the boss activity with a random number. First we will need to create the boss actor. Because there will only ever be one boss alien on screen at any time, we can just use one actor created near the start of our code. In this case we don't need to give it coordinates as we will start the game with the boss actor not being drawn. We write `boss = Actor("boss")`.

▲ Lasers can be very bad for your health. Best to avoid them

08 Keeping the boss in the loop

We want to start the game with the boss not being displayed, so we can add to our `init()` function `boss.active = False` and then in our `draw()` function `if boss.active: boss.draw()`, which will mean the boss will not be drawn until we make it active. In our `update()` function, along with our other functions to update elements, we can call `updateBoss()`. This function will update the coordinates of the boss actor if it is active or, if it is not, check to see if we need to start a new boss flying. See **figure2.py** for the `updateBoss()` function.

09 Did you hear that?

You may have noticed that in **figure2.py** we have an element of Pygame Zero that we have not discussed yet, and that is sound. If we write `sounds.explosion.play()`, then the sound file located at **sounds/explosion.wav** will be played. There are many free sound effects for games on the internet. If you use a downloaded WAV file, make sure that it is fairly small. You can edit WAV sound files with programs like Audacity. We can add sound code to other events in the program in the same way, like when a laser is fired.

10 More about the boss

Staying with **figure2.py**, note how we can use random numbers to decide when the boss becomes active and also when the boss fires a laser. You can change the parameters of the `randint()` function to alter the occurrence of these

events. You can also see that we have a simple path calculating system for the boss to make it move diagonally down the screen. We use the `level` variable to alter aspects of the movement. We treat the boss lasers in the same way as the normal alien lasers, but we need to have a check to see if the boss is hit by a player laser. We do this by adding a check to our `checkPlayerLaserHit()` function.

11 Three strikes and you're out

In the previous episode, the game ended if you were hit by a laser. In this version we have three chances before the game ends, and when it does, we want to display a high score table or leader board to be updated from one player to the next. There are a few considerations to think about here. We need a separate screen for our leader board; we need to get players to enter their name to put against each score and we will have to save the score information. In other programs in this series we have used the variable `gameStatus` to control different screens, so let's bring that back for this program.

12 Screen switching with gameStatus

We will need three states for the `gameStatus` variable. If it is set to 0 then we should display an intro screen where we can get the player to type in their name. If it is set to 1 then we want to run code for playing the game. And if it is set to 2 then we display the leader-board page. Let's first deal with the intro screen. Having set our variable to 0 at the top of the code, we need to add a condition to our `draw()` function: `if gameStatus == 0:`. Then, under that, use `drawCentreText()` to show some intro text and display the `player.name` string. To start with, `player.name` will be blank.

13 A name is just a name

Now to respond to the player typing their name into the intro screen. We will write a very simple input routine and put it in the built-in Pygame Zero function `on_key_down()`. **figure3.py** shows how we do this. With this code, if the player presses a key, the name of the key is added to the `player.name` string unless the key is the **BACKSPACE** key, in which case we remove the last character. Notice the rather cunning way of doing that with `player.name = player.name[:-1]`. We also ignore the **RETURN** key, as we can deal with that in our `update()` function.

figure3.py

```
001.  def on_key_down(key):
002.      global player
003.      if gameStatus == 0 and key.name != "RETURN":
004.          if len(key.name) == 1:
005.              player.name += key.name
006.          else:
007.              if key.name == "BACKSPACE":
008.                  player.name = player.name[:-1]
```

▲ Code for capturing keyboard input for the player to input their name on the introduction screen

14 Game on

When the player has entered their name on the intro screen, all we need to do is detect a press of the **RETURN** key in our `update()` function and we can switch to the game part. We can easily do this by just writing `if gameStatus == 0:` and then under that, `if keyboard.RETURN and player.name != "": gameStatus = 1`. We will also now need to put our main game update code under a condition, `if gameStatus == 1:`. We will also need to have the same condition in the `draw()` function. Once this is done, we have a system for switching from intro screen to game screen.

15 Leader of the pack

So now we come to our leader-board screen. It will be triggered when the player loses the third life. When that happens, we set gameStatus to 2 and put a condition in our `draw()` and `update()` functions to react to that. When we switch to our leader board, we need to display the high score list – so, we can write in our `draw()` function: `if gameStatus == 2: drawHighScore()`. Going back to **figure1.py**, you'll see that we left a section at the end commented out, ready for the leader board. We can now fill this in with some code.

16 If only I learned to read and write

We are going to save all our scores in a file so that we can get them back each time the

figure4.py

```
001.  def readHighScore():
002.      global highScore, score, player
003.      highScore = []
004.      try:
005.          hsFile = open("highscores.txt", "r")
006.          for line in hsFile:
007.              highScore.append(line.rstrip())
008.      except:
009.          pass
010.      highScore.append(str(score)+ " " + player.name)
011.      highScore.sort(key=natural_key, reverse=True)
012.
013.  def natural_key(string_):
014.      return [int(s) if s.isdigit() else s for s in
      re.split(r'(\d+)', string_)]
015.
016.  def writeHighScore():
017.      global highScore
018.      hsFile = open("highscores.txt", "w")
019.      for line in highScore:
020.          hsFile.write(line + "\n")
021.
022.  def drawHighScore():
023.      global highScore
024.      y = 0
025.      screen.draw.text("TOP SCORES", midtop=(400, 30),
      owidth=0.5, ocolor=(255,255,255), color=(0,64,255) ,
      fontsize=60)
026.      for line in highScore:
027.          if y < 400:
028.              screen.draw.text(line, midtop=(400, 100+y),
      owidth=0.5, ocolor=(0,0,255), color=(255,255,0) ,
      fontsize=50)
029.              y += 50
030.      screen.draw.text("Press Escape to play again" ,
      center=(400, 550), owidth=0.5, ocolor=(255,255,255),
      color=(255,64,0) , fontsize=60)
```

▲ Code for reading, writing, sorting, and drawing the high score leader board

game is played. We can use a simple text file for this. When a new score is available, we will have to read the old score list in, add our new score to the list, sort the scores into the correct order, and then save the scores back out to create an updated file. So, the code we need to write in our `update()` function will be to call a `readHighScore()` function, set our `gameStatus` to 2, and call a `writeHighScore()` function.

▲ All the aliens have been destroyed. It's time to move up a level

17 Functions need to function

We have named three functions that need writing in the last couple of steps: `drawHighScore()`, `readHighScore()`, and `writeHighScore()`. Have a look at **figure4.py** to see the code that we need in these functions. The file reading and writing are standard Python functions. When reading, we create a list of entries and add each line to a list. We then sort the list into highest-score-first order. When we write the file, we just write each list item to the file. To draw the leader board, we just run through the high-score list that we have sorted and draw the lines of text to the screen.

18 Sort it out

It's worth mentioning the way we are sorting the high scores. In **figure4.py** we are adding a key sorting method to the list sorting function. We do this because the list is a string but we want to sort by the high score, which is numerical, so we break up the string and convert it to an integer and sort based on that value rather than the string. If we didn't do this and sorted as a string then all the scores starting with 9 would come first, then all the 8s, then all the 7s and so on, with 9000 being shown before 80 000, which would be wrong.

19 Well, that's all folks

That's about all we need for our Pygame Zero PiVaders game other than all the additions that you could make to it. For example, you could have different graphics for each row of aliens. We're sure you can improve on the sounds that we have supplied, and there are many ways that the `level` variable can be worked into the code to make the different levels more difficult or more varied.

pivaders2.py

DOWNLOAD THE FULL CODE:
magpi.cc/pgzero5

```
import pgzrun, math, re, time
from random import randint
player = Actor("player", (400, 550))
boss = Actor("boss")
gameStatus = 0
highScore = []

def draw(): # Pygame Zero draw function
    screen.blit('background', (0, 0))
    if gameStatus == 0: # display the title page
        drawCentreText("PIVADERS\n\n\nType your name then\npress Enter to start\n(arrow keys move, space to fire)")
        screen.draw.text(player.name , center=(400, 500), owidth=0.5, ocolor=(255,0,0), color=(0,64,255) , fontsize=60)
    if gameStatus == 1: # playing the game
        player.image = player.images[math.floor(player.status/6)]
        player.draw()
        if boss.active: boss.draw()
        drawLasers()
        drawAliens()
        drawBases()
        screen.draw.text(str(score) , topright=(780, 10), owidth=0.5, ocolor=(255,255,255), color=(0,64,255) , fontsize=60)
        screen.draw.text("LEVEL " + str(level) , midtop=(400, 10), owidth=0.5, ocolor=(255,255,255), color=(0,64,255) , fontsize=60)
        drawLives()
        if player.status >= 30:
            if player.lives > 0:
                drawCentreText("YOU WERE HIT!\nPress Enter to re-spawn")
            else:
                drawCentreText("GAME OVER!\nPress Enter to continue")
        if len(aliens) == 0 :
            drawCentreText("LEVEL CLEARED!\nPress Enter to go to the next level")
    if gameStatus == 2: # game over show the leaderboard
        drawHighScore()

def drawCentreText(t):
    screen.draw.text(t , center=(400, 300), owidth=0.5, ocolor=(255,255,255), color=(255,64,0) , fontsize=60)

def update(): # Pygame Zero update function
    global moveCounter, player, gameStatus, lasers, level, boss
    if gameStatus == 0:
        if keyboard.RETURN and player.name != "":
            gameStatus = 1
    if gameStatus == 1:
        if player.status < 30 and len(aliens) > 0:
            checkKeys()
            updateLasers()
            updateBoss()
            if moveCounter == 0: updateAliens()
            moveCounter += 1
            if moveCounter == moveDelay:
                moveCounter = 0
            if player.status > 0:
                player.status += 1
                if player.status == 30:
                    player.lives -= 1
        else:
            if keyboard.RETURN:
                if player.lives > 0:
                    player.status = 0
                    lasers = []
                    if len(aliens) == 0:
                        level += 1
                        boss.active = False
                        initAliens()
                        initBases()
                else:
                    readHighScore()
                    gameStatus = 2
                    writeHighScore()
    if gameStatus == 2:
        if keyboard.ESCAPE:
            init()
            gameStatus = 0

def on_key_down(key):
    global player
    if gameStatus == 0 and key.name != "RETURN":
        if len(key.name) == 1:
            player.name += key.name
        else:
            if key.name == "BACKSPACE":
                player.name = player.name[:-1]

def readHighScore():
    global highScore, score, player
    highScore = []
    try:
        hsFile = open("highscores.txt", "r")
        for line in hsFile:
            highScore.append(line.rstrip())
    except:
        pass
    highScore.append(str(score)+ " " + player.name)
    highScore.sort(key=natural_key, reverse=True)

```

```
092. def natural_key(string_):
093.     return [int(s) if s.isdigit() else s for s in
     re.split(r'(\d+)', string_)]
094.
095. def writeHighScore():
096.     global highScore
097.     hsFile = open("highscores.txt", "w")
098.     for line in highScore:
099.         hsFile.write(line + "\n")
100.
101. def drawHighScore():
102.     global highScore
103.     y = 0
104.     screen.draw.text("TOP SCORES", midtop=(400,
     30), owidth=0.5, ocolor=(255,255,255),
     color=(0,64,255) , fontsize=60)
105.     for line in highScore:
106.         if y < 400:
107.             screen.draw.text(line, midtop=(400,
     100+y), owidth=0.5, ocolor=(0,0,255),
     color=(255,255,0) , fontsize=50)
108.             y += 50
109.     screen.draw.text("Press Escape to play
     again" , center=(400, 550), owidth=0.5,
     ocolor=(255,255,255), color=(255,64,0) ,
     fontsize=60)
110.
111. def drawLives():
112.     for l in range(player.lives): screen.
     blit("life", (10+(l*32),10))
113.
114. def drawAliens():
115.     for a in range(len(aliens)): aliens[a].draw()
116.
117. def drawBases():
118.     for b in range(len(bases)): bases[b].
     drawClipped()
119.
120. def drawLasers():
121.     for l in range(len(lasers)): lasers[l].draw()
122.
123. def checkKeys():
124.     global player, score
125.     if keyboard.left:
126.         if player.x > 40: player.x -= 5
127.     if keyboard.right:
128.         if player.x < 760: player.x += 5
129.     if keyboard.space:
130.         if player.laserActive == 1:
131.             sounds.gun.play()
132.             player.laserActive = 0
133.             clock.schedule(makeLaserActive, 1.0)
134.             lasers.append(Actor("laser2",
     (player.x,player.y-32)))
135.             lasers[len(lasers)-1].status = 0
136.             lasers[len(lasers)-1].type = 1
137.             score -= 100
138.
139. def makeLaserActive():
140.     global player
141.     player.laserActive = 1
142.
143. def checkBases():
144.     for b in range(len(bases)):
145.         if l < len(bases):
146.             if bases[b].height < 5:
147.                 del bases[b]
148.
149. def updateLasers():
150.     global lasers, aliens
151.     for l in range(len(lasers)):
152.         if lasers[l].type == 0:
153.             lasers[l].y += 2
154.             checkLaserHit(l)
155.             if lasers[l].y > 600: lasers[l].
     status = 1
156.         if lasers[l].type == 1:
157.             lasers[l].y -= 5
158.             checkPlayerLaserHit(l)
159.             if lasers[l].y < 10: lasers[l].status
     = 1
160.     lasers = listCleanup(lasers)
161.     aliens = listCleanup(aliens)
162.
163. def listCleanup(l):
164.     newList = []
165.     for i in range(len(l)):
166.         if l[i].status == 0: newList.append(l[i])
167.     return newList
168.
169. def checkLaserHit(l):
170.     global player
171.     if player.collidepoint((lasers[l].x,
     lasers[l].y)):
172.         sounds.explosion.play()
173.         player.status = 1
174.         lasers[l].status = 1
175.     for b in range(len(bases)):
176.         if bases[b].collideLaser(lasers[l]):
177.             bases[b].height -= 10
178.             lasers[l].status = 1
179.
180. def checkPlayerLaserHit(l):
181.     global score, boss
182.     for b in range(len(bases)):
183.         if bases[b].collideLaser(lasers[l]):
     lasers[l].status = 1
184.     for a in range(len(aliens)):
185.         if aliens[a].collidepoint((lasers[l].x,
     lasers[l].y)):
186.             lasers[l].status = 1
187.             aliens[a].status = 1
188.             score += 1000
189.     if boss.active:
190.         if boss.collidepoint((lasers[l].x,
     lasers[l].y)):
191.             lasers[l].status = 1
192.             boss.active = 0
```

```
            score += 5000

def updateAliens():
    global moveSequence, lasers, moveDelay
    movex = movey = 0
    if moveSequence < 10 or moveSequence > 30:
movex = -15
    if moveSequence == 10 or moveSequence == 30:
        movey = 40 + (5*level)
        moveDelay -= 1
    if moveSequence >10 and moveSequence < 30:
movex = 15
    for a in range(len(aliens)):
        animate(aliens[a], pos=(aliens[a].x
+ movex, aliens[a].y + movey), duration=0.5,
tween='linear')
        if randint(0, 1) == 0:
            aliens[a].image = "alien1"
        else:
            aliens[a].image = "alien1b"
            if randint(0, 5) == 0:
                lasers.append(Actor("laser1",
(aliens[a].x,aliens[a].y)))
                lasers[len(lasers)-1].status = 0
                lasers[len(lasers)-1].type = 0
                sounds.laser.play()
        if aliens[a].y > 500 and player.status ==
0:
            sounds.explosion.play()
            player.status = 1
            player.lives = 1
    moveSequence +=1
    if moveSequence == 40: moveSequence = 0

def updateBoss():
    global boss, level, player, lasers
    if boss.active:
        boss.y += (0.3*level)
        if boss.direction == 0: boss.x -= (1*
level)
        else: boss.x += (1* level)
        if boss.x < 100: boss.direction = 1
        if boss.x > 700: boss.direction = 0
        if boss.y > 500:
            sounds.explosion.play()
            player.status = 1
            boss.active = False
        if randint(0, 30) == 0:
            lasers.append(Actor("laser1",
(boss.x,boss.y)))
            lasers[len(lasers)-1].status = 0
            lasers[len(lasers)-1].type = 0
    else:
        if randint(0, 800) == 0:
            boss.active = True
            boss.x = 800
            boss.y = 100
            boss.direction = 0

def init():
    global lasers, score, player, moveSequence,
moveCounter, moveDelay, level, boss
    initAliens()
    initBases()
    moveCounter = moveSequence = player.status =
score = player.laserCountdown = 0
    lasers = []
    moveDelay = 30
    boss.active = False
    player.images =
["player","explosion1","explosion2","explosion3",
"explosion4","explosion5"]
    player.laserActive = 1
    player.lives = 3
    player.name = ""
    level = 1

def initAliens():
    global aliens, moveCounter, moveSequence
    aliens = []
    moveCounter = moveSequence = 0
    for a in range(18):
        aliens.append(Actor("alien1", (210+(a %
6)*80,100+(int(a/6)*64))))
        aliens[a].status = 0

def drawClipped(self):
    screen.surface.blit(self._surf, (self.x-32,
self.y-self.height+30),(0,0,64,self.height))

def collideLaser(self, other):
    return (
        self.x-20 < other.x+5 and
        self.y-self.height+30 < other.y and
        self.x+32 > other.x+5 and
        self.y-self.height+30 + self.height >
other.y
    )

def initBases():
    global bases
    bases = []
    bc = 0
    for b in range(3):
        for p in range(3):
            bases.append(Actor("base1",
midbottom=(150+(b*200)+(p*40),520)))
            bases[bc].drawClipped = drawClipped.__
get__(bases[bc])
            bases[bc].collideLaser =
collideLaser.__get__(bases[bc])
            bases[bc].height = 60
            bc +=1

init()
pgzrun.go()
```

Part 05

Pygame Zero
Hungry Pi-Man

Maze games have been popular since the 1980s. Here we will be using more advanced Python programming techniques to create our own addition to the genre

You'll Need

- An image manipulation program such as GIMP, or images available from **magpi.cc/pgzero6**
- The latest version of Pygame Zero
- USB joystick or gamepad (optional)

The concept of Hungry Pi-Man is quite simple. Pi-Man eats green dots (peas) in a maze to score points. Avoid the flames unless you have just eaten a power-up, in which case you can eat them. In this series we have gradually introduced new elements of Pygame Zero and also concepts around writing games. This is the first instalment in a two-part tutorial which will show you some more tricks to writing arcade games with Pygame Zero. We will also use some more advanced programming concepts to make our games even better. In this first part, we will put together the basics of the Hungry Pi-Man game and introduce the concept of adding extra Python modules to our program.

01 Let's get stuck in

As with the more recent episodes of this series, let's jump straight in, assuming that we have our basic Pygame Zero setup done. Let's set our window size to `WIDTH = 600` and `HEIGHT = 660`. This will give us room for a roughly square maze and a header area for some game information. We can get our gameplay area set up straight away by blitting two graphics – 'header' and 'colourmap' – to `0,0` and `0,80` respectively in the `draw()` function. You can make these graphics yourself or you can use ours, which can be found at **magpi.cc/pgzero6**.

02 It's amazing

Our maze for the game has a specific layout, but you can make your own design if you want. If you do make your own, you'll also have to create two more maps (we'll come to those in a bit) which help with the running of the game. The main things about the map is that it has a central area where the flames start from and it doesn't have any other closed-in areas that the flames are likely to get trapped in (they can be a bit stupid sometimes).

▲ Our hungry Pi-Man explores the maze, gobbling green peas while avoiding flames

03 Pie and peas

Our next challenge is to get a player actor moving around the maze. To fit our Hungry Pi-Man theme, for this we will have a hungry pie that goes around eating green peas – yes, it's a rather surreal idea, but no stranger than the themes of many 1980s arcade games!

We'll need two frames for our character: one with the mouth open and one with it closed. We can create our player actor near the top of the code using `player = Actor("piman_o")`. This will create the actor with the mouth-open graphic. We will then set the actor's location in an `init()` function, as in previous programs.

04 Modulify to simplify

We can get our player onto the play area by setting `player.x = 290` and `player.y = 570` in the `init()` function and then call `player.draw()` in the `draw()` function, but to move the player character we'll need to get some input from the player. Previously we have used keyboard and mouse input, but this time we are going to have the option of joystick or gamepad input. Pygame Zero doesn't currently directly support gamepads, but we are going to borrow a bit of the Pygame module to get this working. We are also going to make a separate Python module for our input.

05 It's a joystick.init

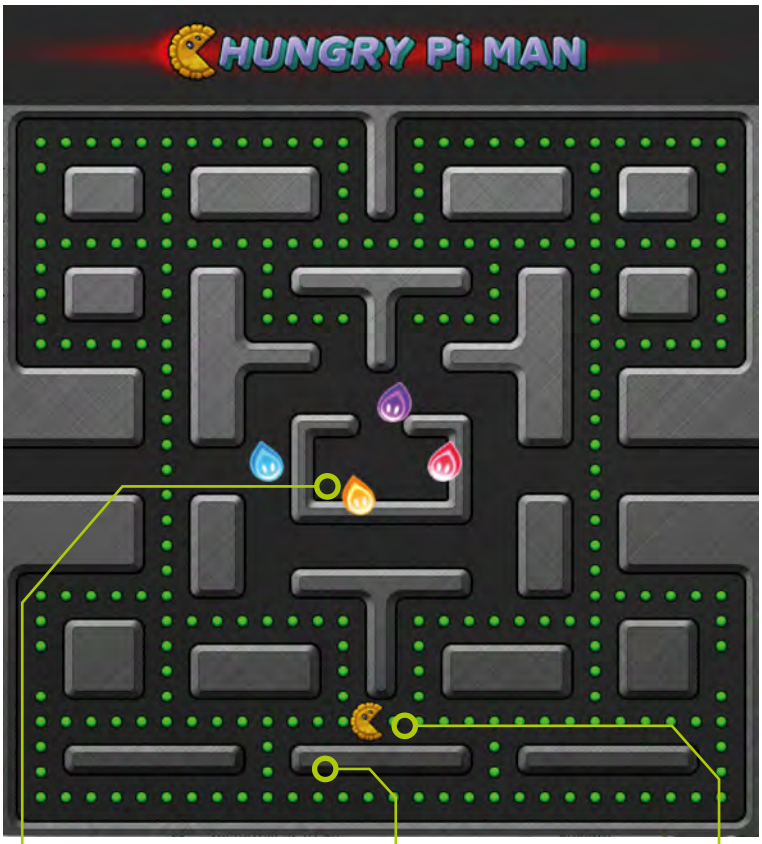
Setting up a new module is easy. All we need to do is make a new file, in this case **gameinput.py**, and in our main program at the top, write `import gameinput`. In this new file we can import the Pygame functions we need with `from pygame import joystick, key` and `from pygame.locals import *`. We can then initialise the Pygame joystick object (this also includes gamepads) by typing `joystick.init()`. We can find out how many joysticks or gamepads are connected by using `joystick_count = joystick.get_count()`. If we find any joysticks connected, we need to initialise them individually – see **figure1.py**.

06 Checking the input

We can now write a function in our gameinput module to check input from the player. If we define the function with `def checkInput(p):` we can get the x axis of a joystick using `joyin.get_axis(0)` and the y axis by using `joyin.get_axis(1)`. The numbers that are returned from these calls will be between -1 and +1, with 0 being the central position. We can check to see if the values are over 0.8 or under -0.8, as, depending on the device, we may not actually see -1 or 1 being returned. You may like to test this with your gamepad or joystick to see what range of values are returned.

07 Up, down, left, or right

The variable `p` that we are passing into our `checkInput()` function will be the player actor. We



The maze is made of corridors and maze walls

Flames move around the maze, looking for Pi-Man

The player is represented by the Pi-Man character that moves around the maze, eating dots

figure1.py

```
001. # gameinput Module
002.
003. from pygame import joystick, key
004. from pygame.locals import *
005.
006. joystick.init()
007. joystick_count = joystick.get_count()
008.
009. if(joystick_count > 0):
010.     joyin = joystick.Joystick(0)
011.     joyin.init()
012.     # For the purposes of this tutorial
013.     # we are only going to use the first
014.     # joystick that is connected.
```

Top Tip

Modules

Using separate modules means not only is your code easier to follow, but it's easier for a team to work on.

figure2.py

```
001. def checkInput(p):
002.     global joyin, joystick_count
003.     xaxis = yaxis = 0
004.     if joystick_count > 0:
005.         xaxis = joyin.get_axis(0)
006.         yaxis = joyin.get_axis(1)
007.     if key.get_pressed()[K_LEFT] or xaxis < -0.8:
008.         p.angle = 180
009.         p.movex = -20
010.     if key.get_pressed()[K_RIGHT] or xaxis > 0.8:
011.         p.angle = 0
012.         p.movex = 20
013.     if key.get_pressed()[K_UP] or yaxis < -0.8:
014.         p.angle = 90
015.         p.movey = -20
016.     if key.get_pressed()[K_DOWN] or yaxis > 0.8:
017.         p.angle = 270
018.         p.movey = 20
```

figure3.py

```
001. # inside update() function
002.
003.     if player.movex or player.movey:
004.         inputLock()
005.         animate(player, pos=(player.x + player.
     movex, player.y + player.movey), duration=1/SPEED,
     tween='linear', on_finished=inputUnLock)
006.
007. # outside update() function
008.
009. def inputLock():
010.     global player
011.     player.inputActive = False
012.
013. def inputUnLock():
014.     global player
015.     player.movex = player.movey = 0
016.     player.inputActive = True
```

can test each of the directions of the joystick at the same time as the keyboard and then set the player angle (so that it points in the correct direction for movement) and also how much it needs to move. We'll set these by saying (for example, if the left arrow is pressed or the joystick is moved to the left) `if key.get_pressed()[K_LEFT] or xaxis < -0.8:` and then `p.angle = 180` and `p.movex = -20`. See **figure2.py** for the full `checkInput()` function.

▲ You can plug a gamepad or joystick into one of the USB ports on your Raspberry Pi

08 Get a move on!

Now we have our input function set up, we can call it from the `update()` function. Because this function is in a different module, we need to prefix it with the module name. In the `update()` function we write `gameinput.checkInput(player)`. After this function has been called, if there has been any input, we should have some variables set in the player actor that we can use to move. We can say `if player.movex or player.movey:` and then use the `animate()` function to move by the amount specified in `player.movex` and `player.movey`.

09 Hold your horses

The way we have the code at the moment means that any time there is some input, we fire off a new animation. This will soon mean that layers of animation get called over the top of each other, but what we want is for the animation to run and then start looking for new input. To do this we need an input locking system. We can call an input lock function before the move and then wait for the animation to finish before unlocking to look for more input. Look at **figure3.py** to see how we can make this locking system.

10 You can't just move anywhere

Now, here comes the interesting bit. We want our player actor to move around the maze, but at the moment it will go though the walls and even off the screen. We need to restrict the movement only to the corridors of the maze. There are several different ways we could do this, but for this game we're going to have an image map marking the

areas that the player actor can move within. The map will be a black and white one, showing just the corridors as black and the walls as white. We will then look at the map in the direction we want to move and see if it is black; if it is, we can move.

11 Testing the map

To be able to test the colour of a part of an image, we need to borrow a few functions from Pygame again. We'll also put our map functions in a separate module. So make a new Python file and call it **gamemaps.py** and in it we'll write `from pygame import image, Color`.

We must also load in our movement map, which we need to do in the Pygame way: `moveimage = image.load('images/pimanmovemap.png')`. Then all we need to do is write a function to check that the direction of the player is valid. See **figure4.py** for this function.

12 Using the movemap

To use this new module, we need to `import gamemaps` at the top of our main code file and then, before we animate the player (but after we have checked for input), we can call `gamemaps.checkMovePoint(player)`,which will zero the `movex` and `movey` variables of the player if the move is not possible. So now we should find that the player actor can only move inside the corridors. We do have one special case that you may have noticed in **figure4.py**, and that is because there is one corridor where the player can move from one side of the screen to the other.

13 You spin me round

There is one more aspect to the movement of the player actor, and that is the animation. As Pi-Man moves, the mouth opens and shuts and points in the direction of the movement. The mouth opening and closing is easy enough: we have an image for open and one for closed and alternate between the two. For pointing in the correct direction, we can rotate the player actor. Unfortunately, this has a slight problem that Pi-Man will be upside-down when moving left. So we just need to have one version that is switched the other way round. See **figure5.py** for a function that sorts out all of this.

figure4.py

```
001. # gamemaps module
002. from pygame import image, Color
003. moveimage = image.load('images/pimanmovemap.png')
004.
005. def checkMovePoint(p):
006.     global moveimage
007.     if p.x+p.movex < 0: p.x = p.x+600
008.     if p.x+p.movex > 600: p.x = p.x-600
009.     if moveimage.get_at((int(p.x+p.movex), int(p.y+
     p.movey-80))) != Color('black'):
010.         p.movex = p.movey = 0
```

figure5.py

```
001. def getPlayerImage():
002.     global player
003.     # we need to import datetime at the top of our code
004.     dt = datetime.now()
005.     a = player.angle
006.     # this next line will give us a number between
007.     # 0 and 5 depending on the time and SPEED
008.     tc = dt.microsecond%(500000/SPEED)/(100000/SPEED)
009.     if tc > 2.5 and (player.movex != 0 or
     player.movey !=0):
010.         # this is for the closed mouth images
011.         if a != 180:
012.             player.image = "piman_c"
013.         else:
014.             # reverse image if facing left
015.             player.image = "piman_cr"
016.     else:
017.         # this is for the open mouth images
018.         if a != 180:
019.             player.image = "piman_o"
020.         else:
021.             player.image = "piman_or"
022.     # set the angle on the player actor
023.     player.angle = a
```

Top Tip

Pygame

Pygame Zero is based on Pygame, but if you want to use some of the Pygame functions, best to do it in a separate module to avoid confusion.

figure6.py

```
# This goes in the main code file.

def initDots():
    global piDots
    piDots = []
    a = x = 0
    while x < 30:
        y = 0
        while y < 29:
            if gamemaps.checkDotPoint(10+x*20, 10+y*20):
                piDots.append(Actor("dot",(10+x*20,
90+y*20)))
                piDots[a].status = 0
                a += 1
            y += 1
        x += 1

# This goes in the gamemaps module file.

dotimage = image.load('images/pimandotmap.png')

def checkDotPoint(x,y):
    global dotimage
    if dotimage.get_at((int(x), int(y))) ==
Color('black'):
        return True
    return False

# This bit goes in the draw() function.

    piDotsLeft = 0
    for a in range(len(piDots)):
        if piDots[a].status == 0:
            piDots[a].draw()
            piDotsLeft += 1
        if piDots[a].collidepoint((player.x, player.y)):
            piDots[a].status = 1
    # if there are no dots left, the player has won
    if piDotsLeft == 0: player.status = 2
```

14 Spot on

So when we have put in a call to `getPlayerImage()` just before we draw the player actor, we should have Pi-Man moving around, chomping and pointing in the correct direction. Now we need something to chomp. We are going to create a set of dots at even spacings along most of the corridors. An easy way to do this is to use a similar technique that we're using for testing where the corridors are. If we make an image map of the places the dots need to go and loop over the whole map, only placing dots where it is black, we can get the desired effect.

15 Tasty, tasty dots

To get our dots doing their thing, we'll need to code a few things. We need to initialise actors for each dot, we need to draw each dot, and if the player eats the dot, we need to stop drawing it; **figure6.py** shows how we can do each of these jobs. We need `initDots()`, we need to add another function to **gamemaps.py** to work out where to position the dots, and we need to add some drawing code to the `draw()` function. In addition to the code in **figure6.py**, we need to add a call to `initDots()` in our `init()` function.

16 Avoid the flames

Now that we have our Pi-Man happily munching green peas, we must introduce our villains to the mix. Four hot flames, each rendered in a different colour, roam the maze looking for Pi-Man, starting from an enclosure in the centre of the map. We can initialise each flame as an actor to appear at the centre of the maze and keep them in a list called `flames[]`. To start off with, we'll just make them move around randomly. The way we can do this is to set a random direction (`flames[g].dir`) for each and then keep them moving until they hit a wall.

17 Random motion

We can use the same system that we used to check player movement for the flames. Each time we move a flame – `moveFlames()` – we can get a list of which directions are available to it. If the current direction (`flames[g].dir`) is not available, then we randomly pick another direction until we find one that we can move in. We can also have a random occurrence of changing direction, just to make it a bit less predictable – and if the flames collide with each other, we could do the same. When we have moved the flames with the `animate()` function, we get it to count how many flames have finished moving. When they are all done, we can call the `moveFlames()` function again.

Top Tip

Animations

When using the `animate()` function, it is best to use the callback function to see when it has finished, as different systems may work at different speeds.

18 Light a flame

The last thing to do with our flames is to actually draw them to the screen. We can create a function called `drawFlames()` where we loop through the four flames and draw them to the screen. One of the details of the original game was that the eyes of the flames would follow the player; we can do this by setting the flame image to reverse if the player is to the left of the flame. We have numbered images so that flame one is **flame1.png** and flame two is **flame2.png**, etc. Have a look at the full **piman1.py** program listing to see all the functions that make the flames work.

19 Game over

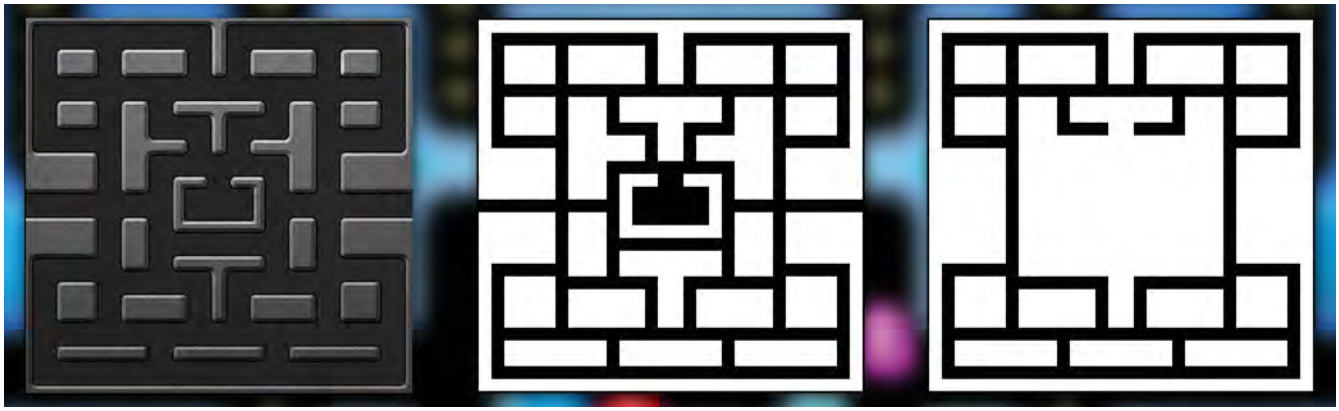
Of course, we need to deal with the end-of-the-game conditions and, as before, we can use a status variable. In this case we have previously set `player.status = 2` if the player wins. We can check to see if a flame collides with the player and set `player.status = 1`. Then we just need to display some text in the `draw()` function based on this variable. And that's it for part one. In the next part we'll be giving the flames more brains, adding levels, lives, and power-ups – and adding some sweet, soothing music and sound effects.

gameinput.py

Language: **Python 3**

```
001. # gameinput Module
002.
003. from pygame import joystick, key
004. from pygame.locals import *
005.
006. joystick.init()
007. joystick_count = joystick.get_count()
008.
009. if(joystick_count > 0):
010.     joyin = joystick.Joystick(0)
011.     joyin.init()
012.
013. def checkInput(p):
014.     global joyin, joystick_count
015.     xaxis = yaxis = 0
016.     if joystick_count > 0:
017.         xaxis = joyin.get_axis(0)
018.         yaxis = joyin.get_axis(1)
019.     if key.get_pressed()[K_LEFT] or xaxis < -0.8:
020.         p.angle = 180
021.         p.movex = -20
022.     if key.get_pressed()[K_RIGHT] or xaxis > 0.8:
023.         p.angle = 0
024.         p.movex = 20
025.     if key.get_pressed()[K_UP] or yaxis < -0.8:
026.         p.angle = 90
027.         p.movey = -20
028.     if key.get_pressed()[K_DOWN] or yaxis > 0.8:
029.         p.angle = 270
030.         p.movey = 20
```

Three maps are used: one which we see, one to check possible movements, and one to check where dots are to be placed



Colour Map | Movement Map | Dot Location Map

gamemaps.py

Language: **Python 3**

```
# gamemaps module

from pygame import image, Color
moveimage = image.load('images/pimanmovemap.png')
dotimage = image.load('images/pimandotmap.png')

def checkMovePoint(p):
    global moveimage
    if p.x+p.movex < 0: p.x = p.x+600
    if p.x+p.movex > 600: p.x = p.x-600
    if moveimage.get_at((int(p.x+p.movex), int(p.y+p.movey-80))) != Color('black'):
        p.movex = p.movey = 0

def checkDotPoint(x,y):
    global dotimage
    if dotimage.get_at((int(x), int(y))) == Color('black'):
        return True
    return False

def getPossibleDirection(g):
    global moveimage
    if g.x-20 < 0:
        g.x = g.x+600
    if g.x+20 > 600:
        g.x = g.x-600
    directions = [0,0,0,0]
    if g.x+20 < 600:
        if moveimage.get_at((int(g.x+20), int(g.y-80))) == Color('black'): directions[0] = 1
    if g.x < 600 and g.x >= 0:
        if moveimage.get_at((int(g.x), int(g.y-60))) == Color('black'): directions[1] = 1
    if g.x-20 >= 0:
        if moveimage.get_at((int(g.x-20), int(g.y-80))) == Color('black'): directions[2] = 1
    if g.x < 600 and g.x >= 0:
        if moveimage.get_at((int(g.x), int(g.y-100))) == Color('black'): directions[3] = 1
    return directions
```

piman1.py

Language: **Python 3**

```
import pgzrun
import gameinput
import gamemaps
from random import randint
from datetime import datetime
WIDTH = 600
HEIGHT = 660

player = Actor("piman_o") # Load in the player Actor image
SPEED = 3

def draw(): # Pygame Zero draw function
    global piDots, player
    screen.blit('header', (0, 0))
    screen.blit('colourmap', (0, 80))
    piDotsLeft = 0
    for a in range(len(piDots)):
        if piDots[a].status == 0:
            piDots[a].draw()
            piDotsLeft += 1
        if piDots[a].collidepoint((player.x, player.y)):
            piDots[a].status = 1
    if piDotsLeft == 0: player.status = 2
    drawFlames()
    getPlayerImage()
    player.draw()
    if player.status == 1: screen.draw.text("GAME OVER", center=(300, 434), owidth=0.5, ocolor=(255,255,255), color=(255,64,0) , fontsize=40)
    if player.status == 2: screen.draw.text("YOU WIN!", center=(300, 434), owidth=0.5, ocolor=(255,255,255), color=(255,64,0) , fontsize=40)

def update(): # Pygame Zero update function
    global player, moveFlamesFlag, flames
    if player.status == 0:
        if moveFlamesFlag == 4: moveFlames()
        for g in range(len(flames)):
            if flames[g].collidepoint((player.x, player.y)):
                player.status = 1
                pass
        if player.inputActive:
            gameinput.checkInput(player)
            gamemaps.checkMovePoint(player)
            if player.movex or player.movey:
                inputLock()
                animate(player, pos=(player.x + player.movex, player.y + player.movey), duration=1/SPEED, tween='linear', on_finished=inputUnLock)

def init():
```

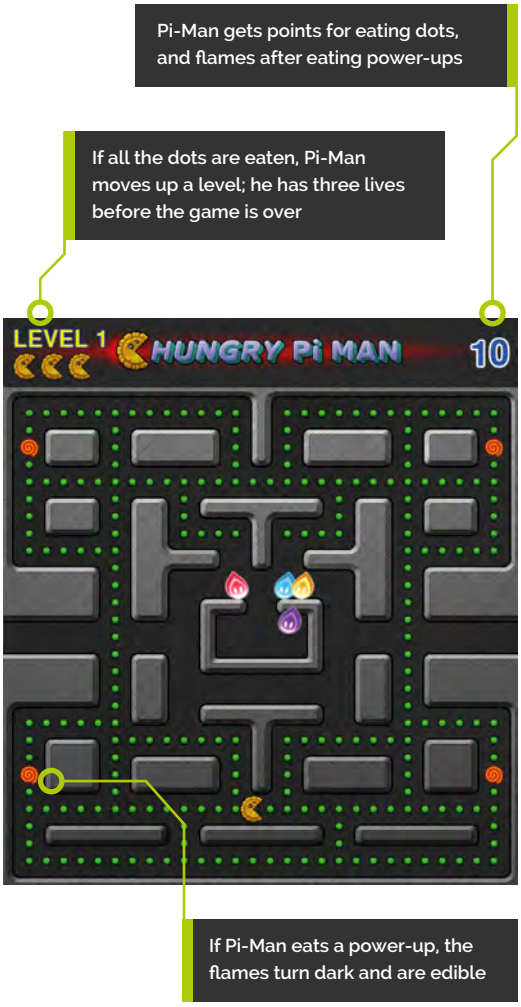
DOWNLOAD THE FULL CODE:
magpi.cc/pgzero6

```
    global player
    initDots()
    initFlames()
    player.x = 290
    player.y = 570
    player.status = 0
    inputUnLock()

def getPlayerImage():
    global player
    dt = datetime.now()
    a = player.angle
    tc = dt.microsecond%(500000/SPEED)/(100000/SPEED)
    if tc > 2.5 and (player.movex != 0 or player.movey !=0):
        if a != 180:
            player.image = "piman_c"
        else:
            player.image = "piman_cr"
    else:
        if a != 180:
            player.image = "piman_o"
        else:
            player.image = "piman_or"
    player.angle = a

def drawFlames():
    for g in range(len(flames)):
        if flames[g].x > player.x:
            flames[g].image = "flame"+str(g+1)+"r"
        else:
            flames[g].image = "flame"+str(g+1)
        flames[g].draw()

def moveFlames():
    global moveFlamesFlag
    dmoves = [(1,0),(0,1),(-1,0),(0,-1)]
    moveFlamesFlag = 0
    for g in range(len(flames)):
        dirs = gamemaps.getPossibleDirection(flames[g])
        if flameCollided(flames[g],g) and randint(0,3) == 0: flames[g].dir = 3
        if dirs[flames[g].dir] == 0 or randint(0,50) == 0:
            d = -1
            while d == -1:
                rd = randint(0,3)
                if dirs[rd] == 1:
                    d = rd
            flames[g].dir = d
        animate(flames[g], pos=(flames[g].x + dmoves[flames[g].dir][0]*20, flames[g].y + dmoves[flames[g].dir][1]*20), duration=1/SPEED, tween='linear', on_finished=flagMoveFlames)

def flagMoveFlames():
    global moveFlamesFlag
    moveFlamesFlag += 1

def flameCollided(ga,gn):
    for g in range(len(flames)):
        if flames[g].colliderect(ga) and g != gn:
            return True
    return False

def initDots():
    global piDots
    piDots = []
    a = x = 0
    while x < 30:
        y = 0
        while y < 29:
            if gamemaps.checkDotPoint(10+x*20, 10+y*20):
                piDots.append(Actor("dot",(10+x*20, 90+y*20)))
                piDots[a].status = 0
                a += 1
            y += 1
        x += 1

def initFlames():
    global flames, moveFlamesFlag
    moveFlamesFlag = 4
    flames = []
    g = 0
    while g < 4:
        flames.append(Actor("flame"+str(g+1),(270+(g*20), 370)))
        flames[g].dir = randint(0, 3)
        g += 1

def inputLock():
    global player
    player.inputActive = False

def inputUnLock():
    global player
    player.movex = player.movey = 0
    player.inputActive = True

init()
pgzrun.go()
```

Part 06

Pygame Zero
Hungry Pi-Man: part 2

In part two of our tutorial, we add some groovy features to the basic game created last time, including better enemy AI, power-ups, levels, and sound



You'll Need

- An image manipulation program such as GIMP, or images available from **magpi.cc/pgzero7**
- The latest version of Pygame Zero (1.2)
- USB joystick or gamepad (optional)
- Headphones or speakers

In part one, we created a maze for our player to move around, and restricted movement to just the corridors. We provided some dots (green peas) to eat and some flames to avoid. In this part we are going to give the flames some more brains so that they are a bit more challenging to the player. We will also add the bonus power-ups which turn the flames into tasty edibles, give Pi-Man some extra levels to explore and some extra lives. So far in this series we have not dealt with music, so we will have a go at putting some music and sound effects into the game.

01 Need more brains

Also in part one, we left our flames wandering around the maze randomly without much thought for what they were doing, which was a bit unfair as Pi-Man could evade them without too much trouble. In the original game, each flame had a program that it followed to characterise its movements. We are going to add some brains to two of the flames. The first we will make follow Pi-Man, and the second we will get to ambush by moving ahead of Pi-Man. We will still leave in some random movement, otherwise it may get a bit too difficult.

02 Follow the leader

First, let's get the red flame to follow Pi-Man. We already have a `moveFlames()` function

Adding a brain to a flame to follow the player

from part one and we can add a condition to see if we are dealing with the first flame: `if g == 0: followPlayer(g, dirs)`. This calls `followPlayer()` if it's the first flame. The `followPlayer()` function receives a list of directions that the flame can move in. It then tests the x coordinate of the player against the x coordinate of the flame and, if the direction is valid, sets the flame direction to move toward the player. Then it does the same with the y coordinates.

03 Y over x

The keen-witted among you will have noticed that if x and y movements towards the player are both valid, then the y direction will always win. We could throw in another random number to choose between the two, but in testing this arrangement it doesn't cause any significant problem with the movement. See **figure1.py** for the `followPlayer()` function. You will see there is a special condition `aboveCentre()` when we check the downward movement. We are checking that the flame is not just above the centre, otherwise it will go back into its starting enclosure.

04 The central problem

If we go back to the `moveFlames()` function, we need another centre-related condition: `if inTheCentre(flames[g])`. This is because if we leave the flame to randomly move around our centre enclosure, it may take a long time to get out. In part one, you may have noticed that from time to time one flame would get stuck in the centre. What we do is, if we detect that a flame is in the centre, we always default to direction 3, which is up. If we run the game with this condition and the `followPlayer()` function, we should see all the flames making their way straight out of the centre and then the red flame making a bee-line towards Pi-Man.

figure1.py

```
001. def followPlayer(g, dirs):
002.     d = flames[g].dir
003.     if d == 1 or d == 3:
004.         if player.x > flames[g].x and dirs[0] == 1:
005.             flames[g].dir = 0
006.         if player.x < flames[g].x and dirs[2] == 1:
007.             flames[g].dir = 2
008.     if d == 0 or d == 2:
009.         if player.y > flames[g].y and dirs[1] == 1 and not
     aboveCentre(flames[g]): flames[g].dir = 1
010.         if player.y < flames[g].y and dirs[3] == 1:
011.             flames[g].dir = 3
012.
013.
014. def aboveCentre(ga):
015.     if ga.x > 220 and ga.x < 380 and ga.y > 300 and ga.y
     < 320:
016.         return True
017.     return False
```

figure2.py

```
# This code goes in the update() function

    if player.status == 1:
        i = gameinput.checkInput(player)
        if i == 1:
            player.status = 0
            player.x = 290
            player.y = 570

# This code goes in the gameinput module
# in the checkInput() function

    if joystick_count > 0:
        jb = joyin.get_button(1)
    else:
        jb = 0
    if p.status == 1:
        if key.get_pressed()[K_RETURN] or jb:
            return 1
```

05 It's an ambush!

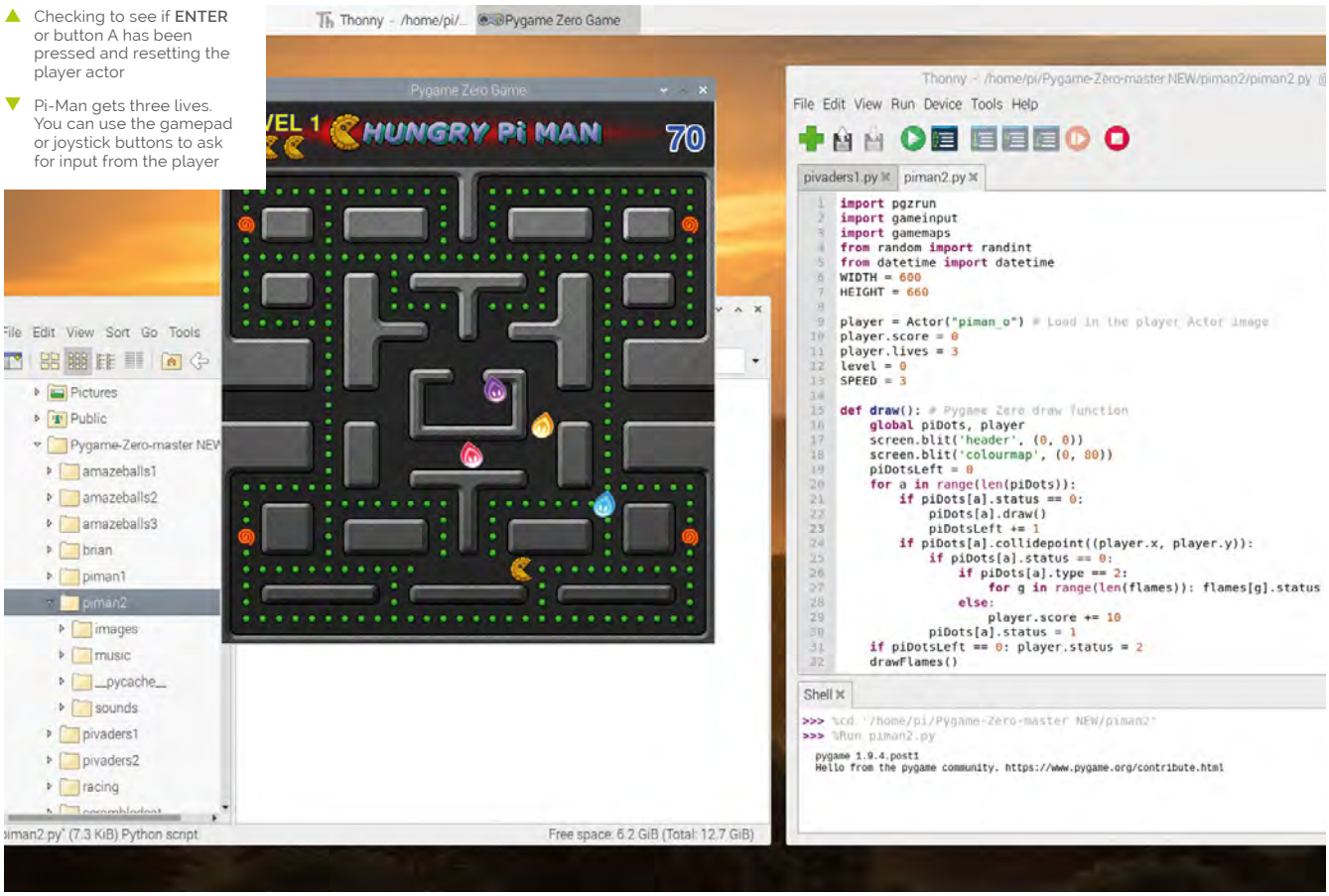
So, the next brain to implant is for the second flame. We will add a function `ambushPlayer()` in the same way we did for the first flame, but this time `if g == 1:`. The `ambushPlayer()` function works very much like the `followPlayer()` function, but this time we just check the direction that Pi-Man is currently moving and try to move in that direction. We, of course, cannot know which direction the player is going to move, and this may seem a bit of a simplistic approach to ambushing the player, but it is surprising how many times Pi-Man ends up wedged between these two flames with this method.

06 Scores on the doors

Brain functions could be added to all the flames, but we are going to leave the flame brains for now as there is plenty more to do to get our

▲ Checking to see if **ENTER** or button A has been pressed and resetting the player actor

▼ Pi-Man gets three lives. You can use the gamepad or joystick buttons to ask for input from the player



game completed. Before we go any further, we ought to get a scoring system going and reward Pi-Man for all the dots eaten. We can attach the score variable to the player actor near the top of our code with `player.score = 0` and then each time a dot is eaten we add 10 to the score with `player.score += 10`. We can also display the score in the `draw()` function (probably top right is best) with `screen.draw.text()`.

07 Three strikes and you're out!

As is the tradition in arcade games, you get three lives before it's game over. If you followed our previous tutorial for PiVaders, you will already know how we do this. We just add a lives variable to the player actor and then each time Pi-Man is caught by a flame, we take a life off, set `player.status = 1`, and print a message to say press **ENTER**. When pressed, we set `player.status = 0` and send Pi-Man back to the starting place. Then we continue. Have a look at **figure2.py** to see the code we add to reset Pi-Man to the start.

08 Printing lives

We have the system for keeping track of the `player.lives` variable, but we also need to show the player how many lives they have left. We can do this with a simple loop like we used in the previous PiVaders tutorial. We can have a `drawLives()` function which we call from our `draw()` function. In that function, we go round a loop for the number of lives we have by saying `for l in range(player.lives):` and then we can use the same image that we use for the player and say `screen.blit("piman_o", (10+(l*32),40))`.

09 Which button to press

You may notice in **figure2.py** that in our gameinput module we are checking a joystick button as well as the **ENTER** key. You may want to do a few tests with the gamepads or joysticks that you're using, as the buttons may have different numbers. You can also prompt the player to press (in this case) the A button to continue. If you were designing a game that relied on several buttons being used, you might want to set up a way of mapping the buttons to values depending on what type of gamepad or joystick is being used.

figure3.py

```
001. # This code is in our main code file (piman2.py)
002.
003. def initDots():
004.     global piDots
005.     piDots = []
006.     a = x = 0
007.     while x < 30:
008.         y = 0
009.         while y < 29:
010.             d = gamemaps.checkDotPoint(10+x*20, 10+y*20)
011.             if d == 1:
012.                 piDots.append(Actor("dot",(10+x*20,
    90+y*20)))
013.                 piDots[a].status = 0
014.                 piDots[a].type = 1
015.                 a += 1
016.             if d == 2:
017.                 piDots.append(Actor("power",(10+x*20,
    90+y*20)))
018.                 piDots[a].status = 0
019.                 piDots[a].type = 2
020.                 a += 1
021.             y += 1
022.         x += 1
023.
024. # This code is in the gamemaps module
025.
026. def checkDotPoint(x,y):
027.     global dotimage
028.     if dotimage.get_at((int(x), int(y))) ==
    Color('black'):
029.         return 1
030.     if dotimage.get_at((int(x), int(y))) == Color('red'):
031.         return 2
032.     return False
```

▲ Updated code to include the creation of power-ups

10 I have the power!

The next item on our list is power-ups. These are large glowing dots that, when eaten, turn all the flames dark. In their dark form they can be eaten for bonus points and they return to the centre of the maze. First, let's devise a way to place the power-ups in the maze. We have updated the **pimandotmap.png** image to include some red squares, instead of black, in the positions where we want our power-ups to be. Then, when we initialise our dots and call `checkDotPoint(x,y)`, we look for red as well as black – **figure3.py** shows how we change our code to do this.

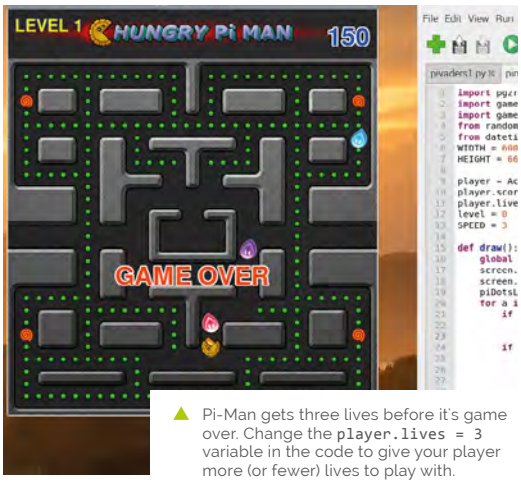
gamemaps.py

> Language: **Python 3**

```
001. from pygame import image, surface, Color
002. moveimage = image.load('images/pimanmovemap.png')
003. dotimage = image.load('images/pimandotmap.png')
004.
005. def checkMovePoint(p):
006.     global moveimage
007.     if p.x+p.movex < 0: p.x = p.x+600
008.     if p.x+p.movex > 600: p.x = p.x-600
009.     if moveimage.get_at((int(p.x+p.movex), int(p.y+
     p.movey-80))) != Color('black'):
010.         p.movex = p.movey = 0
011.
012. def checkDotPoint(x,y):
013.     global dotimage
014.     if dotimage.get_at((int(x), int(y))) ==
     Color('black'):
015.         return 1
016.     if dotimage.get_at((int(x), int(y))) ==
     Color('red'):
017.         return 2
018.     return False
019.
020. def getPossibleDirection(g):
021.     global moveimage
022.     if g.x-20 < 0:
023.         g.x = g.x+600
024.     if g.x+20 > 600:
025.         g.x = g.x-600
026.     directions = [0,0,0,0]
027.     if g.x+20 < 600:
028.         if moveimage.get_at((int(g.x+20),
     int(g.y-80))) == Color('black'): directions[0] = 1
029.     if g.x < 600 and g.x >= 0:
030.         if moveimage.get_at((int(g.x), int(g.y-60)))
     == Color('black'): directions[1] = 1
031.     if g.x-20 >= 0:
032.         if moveimage.get_at((int(g.x-20),
     int(g.y-80))) == Color('black'): directions[2] = 1
033.     if g.x < 600 and g.x >= 0:
034.         if moveimage.get_at((int(g.x), int(g.y-100)))
     == Color('black'): directions[3] = 1
035.     return directions
```

11 Not all dots are the same

We now have a system to place our power-ups in the maze. The next thing to do is to change what happens when Pi-Man eats a power-up compared to a normal dot. At the moment we just add ten points to the player's score if a dot is eaten, so we need to add more code to handle the event of a power-up being eaten. In the `draw()` function, where we look to see if the player has collided with a dot using `collidepoint()`, we then check the status of the dot (to make sure it's still there) and after this we can add a new condition: `if piDots[a].type == 2:`.



▲ Pi-Man gets three lives before it's game over. Change the `player.lives = 3` variable in the code to give your player more (or fewer) lives to play with.

12 High status flames

As we have determined that we are dealing with a power-up (type 2), we can add a loop that goes through the list of flames and changes the status of the flame. Normally the status for a flame is 0. What we are going to do is change the status to a fairly high number (try 1200 to start with). This will indicate that the flames are in their alternate state and we will use the status as a countdown. We will decrement this value each time `update()` is called; when it reaches 0, the flames will turn back to normal.

13 Why so dark?

To make our flame turn dark, we are going to add some conditions to our `drawFlames()` function. We want them to be dark when the status is more than 0, but just to make it interesting we will make them flash when they are about to turn back. So we can write `if flames[g].status > 200 or (flames[g].status > 1 and flames[g].status%2 == 0): flames[g].image = "flame"+str(g+1)+"-"`. What this is saying is that if the status is over 200 then make the flame dark, but if it's less that 200 but greater than 1 then make it dark every other frame. We then have an `else` condition underneath that will set the image to its normal colour.

gameinput.py

Language: **Python 3**

```
001. from pygame import joystick, key
002. from pygame.locals import *
003.
004. joystick.init()
005. joystick_count = joystick.get_count()
006.
007. if(joystick_count > 0):
008.     joyin = joystick.Joystick(0)
009.     joyin.init()
010.
011. def checkInput(p):
012.     global joyin, joystick_count
013.     xaxis = yaxis = 0
014.     if p.status == 0:
015.         if joystick_count > 0:
016.             xaxis = joyin.get_axis(0)
017.             yaxis = joyin.get_axis(1)
018.         if key.get_pressed()[K_LEFT] or xaxis < -0.8:
019.             p.angle = 180
020.             p.movex = -20
021.         if key.get_pressed()[K_RIGHT] or xaxis > 0.8:
022.             p.angle = 0
023.             p.movex = 20
024.         if key.get_pressed()[K_UP] or yaxis < -0.8:
025.             p.angle = 90
026.             p.movey = -20
027.         if key.get_pressed()[K_DOWN] or yaxis > 0.8:
028.             p.angle = 270
029.             p.movey = 20
030.     if joystick_count > 0:
031.         jb = joyin.get_button(1)
032.     else:
033.         jb = 0
034.     if p.status == 1:
035.         if key.get_pressed()[K_RETURN] or jb:
036.             return 1
037.     if p.status == 2:
038.         if key.get_pressed()[K_RETURN] or jb:
039.             return 1
```

14 The tables have turned

Now we have our flames all turning dark when a power-up is eaten, we need to change what happens when Pi-Man collides with them. Instead of taking a life from the `player.lives` variable, we are going to add to the `player.score` variable and send the flame back to the centre. So, the first job is to add a condition in `update()` when we check the flame `collidepoint()` with the player, which would be `if flames[g].status > 0:`. We then add 100 to the `player.score` and `animate()` the flame back to the centre. See **figure4.py** for the updated code.

15 Back to the start

You will notice that when Pi-Man comes into contact with a dark flame, we just animate the actor straight back to the centre in the same time that we normally animate a flame from one position to the next. This is so that we don't hold up the animation on the other flames waiting for the eaten one to get back to the centre. In the original game, the flames would turn into a pair of eyes and then make their way back to the centre along the corridors, but that would take too much extra code for this tutorial.

figure4.py

```
001. # This code is in the update() function
002.
003.         for g in range(len(flames)):
004.             if flames[g].status > 0: flames[g].status -= 1
005.             if flames[g].collidepoint((player.x,
     player.y)):
006.                 if flames[g].status > 0:
007.                     player.score += 100
008.                     animate(flames[g], pos=(290, 370),
     duration=1/SPEED, tween='linear',
     on_finished=flagMoveFlames)
009.                 else:
010.                     player.lives -= 1
011.                     if player.lives == 0:
012.                         player.status = 3
013.                     else:
014.                         player.status = 1
```

▲ Updated flame collision code to send them back to the centre if Pi-Man eats them

16 Time for some music

So far in this series, we have not covered adding music to games. In the documentation of Pygame Zero, music is labelled as experimental, so we will just have to try it out and see what

happens. In the sample GitHub files for this tutorial, there is a directory called **music** and in that directory is an MP3 file that we can use as eighties arcade game background music. To start our music, all we need to do is write `music.play("pm1")` in our `init()` function to start the **music/pm1.mp3** file. You may also want to set the volume with `music.set_volume(0.3)`.

17 More sound effects

The MP3 file will continue playing in a loop until we stop it, so when the game is over (`player.lives = 0`) we can fade the music out with `music.fadeout(3)`. At this stage we can also add some sound effects for when Pi-Man is eating dots. We have a sound in our **sounds** directory called **pi1.mp3** which we will use for this purpose and we can add a line of code just before we animate the player: `sounds.pi1.play()`. This will play the sound every time Pi-Man moves. We can do the same with **pi2.mp3** when a life is lost.

18 Level it up

The last thing we need to put into our game is to allow the player to progress to the next level when all the dots have been eaten. We could incorporate several things to make each level harder, but for the moment let's concentrate on resetting the screen and changing the level. If we define our level variable near the top of our code as `level = 0`, then inside our `init()` function we say `level += 1`, then each time we call `init()` we will increase our level variable. This means that instead of saying that the player has won, we just prompt them to continue, and call `init()` to reset everything and level up.

19 So much to do

The Pi-Man game has many more things that can be added to it. For instance, you could include bonus fruits to collect, the flames might move faster as the levels continue, there could be animations between some of the levels, and the power-ups might run out quicker. You could add all of these things to this game, but we will have to leave you to do that yourself. In the next tutorial, we'll be starting a new Pygame Zero game with isometric 3D graphics.

piman2.py

> Language: **Python 3**

```
001. import pgzrun
002. import gameinput
003. import gamemaps
004. from random import randint
005. from datetime import datetime
006. WIDTH = 600
007. HEIGHT = 660
008.
009. player = Actor("piman_o") # Load in the player Actor image
010. player.score = 0
011. player.lives = 3
012. level = 0
013. SPEED = 3
014.
015. def draw(): # Pygame Zero draw function
016.     global piDots, player
017.     screen.blit('header', (0, 0))
018.     screen.blit('colourmap', (0, 80))
019.     piDotsLeft = 0
020.     for a in range(len(piDots)):
021.         if piDots[a].status == 0:
022.             piDots[a].draw()
023.             piDotsLeft += 1
024.         if piDots[a].collidepoint((player.x, player.y)):
025.             if piDots[a].status == 0:
026.                 if piDots[a].type == 2:
027.                     for g in range(len(flames)): flames[g].status = 1200
028.                 else:
029.                     player.score += 10
030.             piDots[a].status = 1
031.     if piDotsLeft == 0: player.status = 2
032.     drawFlames()
033.     getPlayerImage()
034.     player.draw()
035.     drawLives()
036.     screen.draw.text("LEVEL "+str(level) , topleft=(10, 10), owidth=0.5,
    ocolor=(0,0,255), color=(255,255,0) , fontsize=40)
037.     screen.draw.text(str(player.score) , topright=(590, 20), owidth=0.5,
    ocolor=(255,255,255), color=(0,64,255) , fontsize=60)
038.     if player.status == 3: drawCentreText("GAME OVER")
039.     if player.status == 2: drawCentreText(
    "LEVEL CLEARED!\nPress Enter or Button A\nto Continue")
040.     if player.status == 1: drawCentreText(
    "CAUGHT!\nPress Enter or Button A\nto Continue")
041.
042. def drawCentreText(t):
043.     screen.draw.text(t , center=(300, 434), owidth=0.5,
    ocolor=(255,255,255), color=(255,64,0) , fontsize=60)
044.
045. def update(): # Pygame Zero update function
046.     global player, moveFlamesFlag, flames
```

DOWNLOAD THE FULL CODE:
magpi.cc/pgzero7

```
    if player.status == 0:
        if moveFlamesFlag == 4: moveFlames()
        for g in range(len(flames)):
            if flames[g].status > 0: flames[g].status -= 1
            if flames[g].collidepoint((player.x,
player.y)):
                if flames[g].status > 0:
                    player.score += 100
                    animate(flames[g], pos=(290, 370),
duration=1/SPEED, tween='linear',
on_finished=flagMoveFlames)
                else:
                    player.lives -= 1
                    sounds.pi2.play()
                    if player.lives == 0:
                        player.status = 3
                        music.fadeout(3)
                    else:
                        player.status = 1
        if player.inputActive:
            gameinput.checkInput(player)
            gamemaps.checkMovePoint(player)
            if player.movex or player.movey:
                inputLock()
                sounds.pi1.play()
                animate(player, pos=(player.x +
player.movex, player.y + player.movey), duration=1/SPEED,
tween='linear', on_finished=inputUnLock)
    if player.status == 1:
        i = gameinput.checkInput(player)
        if i == 1:
            player.status = 0
            player.x = 290
            player.y = 570
    if player.status == 2:
        i = gameinput.checkInput(player)
        if i == 1:
            init()

def init():
    global player, level
    initDots()
    initFlames()
    player.x = 290
    player.y = 570
    player.status = 0
    inputUnLock()
    level += 1
    music.play("pm1")
    music.set_volume(0.2)

def drawLives():
    for l in range(player.lives): screen.blit("piman_o",
(10+(l*32),40))

def getPlayerImage():
    global player
    dt = datetime.now()
    a = player.angle
    tc = dt.microsecond%(500000/SPEED)/(100000/SPEED)
    if tc > 2.5 and (player.movex != 0 or player.movey
!=0):
        if a != 180:
            player.image = "piman_c"
        else:
            player.image = "piman_cr"
    else:
        if a != 180:
            player.image = "piman_o"
        else:
            player.image = "piman_or"
    player.angle = a

def drawFlames():
    for g in range(len(flames)):
        if flames[g].x > player.x:
            if flames[g].status > 200 or (flames[g].status
> 1 and flames[g].status%2 == 0):
                flames[g].image = "flame"+str(g+1)+"-"
            else:
                flames[g].image = "flame"+str(g+1)+"r"
        else:
            if flames[g].status > 200 or (flames[g].status
> 1 and flames[g].status%2 == 0):
                flames[g].image = "flame"+str(g+1)+"-"
            else:
                flames[g].image = "flame"+str(g+1)
        flames[g].draw()

def moveFlames():
    global moveFlamesFlag
    dmoves = [(1,0),(0,1),(-1,0),(0,-1)]
    moveFlamesFlag = 0
    for g in range(len(flames)):
        dirs = gamemaps.getPossibleDirection(flames[g])
        if inTheCentre(flames[g]):
            flames[g].dir = 3
        else:
            if g == 0: followPlayer(g, dirs)
            if g == 1: ambushPlayer(g, dirs)

        if dirs[flames[g].dir] == 0 or randint(0,50) == 0:
```

```
            d = -1
            while d == -1:
                rd = randint(0,3)
                if aboveCentre(flames[g]) and rd == 1:
                    rd = 0
                if dirs[rd] == 1:
                    d = rd
            flames[g].dir = d
        animate(flames[g], pos=(flames[g].x
+ dmoves[flames[g].dir][0]*20, flames[g].y +
dmoves[flames[g].dir][1]*20), duration=1/SPEED,
tween='linear', on_finished=flagMoveFlames)

def followPlayer(g, dirs):
    d = flames[g].dir
    if d == 1 or d == 3:
        if player.x > flames[g].x and dirs[0] == 1:
flames[g].dir = 0
        if player.x < flames[g].x and dirs[2] == 1:
flames[g].dir = 2
    if d == 0 or d == 2:
        if player.y > flames[g].y and dirs[1] == 1 and not
aboveCentre(flames[g]): flames[g].dir = 1
        if player.y < flames[g].y and dirs[3] == 1:
flames[g].dir = 3


def ambushPlayer(g, dirs):
    d = flames[g].dir
    if player.movex > 0 and dirs[0] == 1: flames[g].dir = 0
    if player.movex < 0 and dirs[2] == 1: flames[g].dir = 2

    if player.movey > 0 and dirs[1] == 1 and not
aboveCentre(flames[g]): flames[g].dir = 1
    if player.movey < 0 and dirs[3] == 1: flames[g].dir = 3

def inTheCentre(ga):
    if ga.x > 220 and ga.x < 380 and ga.y > 320 and ga.y <
420:
        return True
    return False

def aboveCentre(ga):
    if ga.x > 220 and ga.x < 380 and ga.y > 300 and ga.y <
320:
        return True
    return False

def flagMoveFlames():
    global moveFlamesFlag
    moveFlamesFlag += 1

def flameCollided(ga,gn):
    for g in range(len(flames)):
        if flames[g].colliderect(ga) and g != gn:
            return True
    return False

def initDots():
    global piDots
    piDots = []
    a = x = 0
    while x < 30:
        y = 0
        while y < 29:
            d = gamemaps.checkDotPoint(10+x*20, 10+y*20)
            if d == 1:
                piDots.append(Actor("dot",(10+x*20,
90+y*20)))
                piDots[a].status = 0
                piDots[a].type = 1
                a += 1
            if d == 2:
                piDots.append(Actor("power",(10+x*20,
90+y*20)))
                piDots[a].status = 0
                piDots[a].type = 2
                a += 1
            y += 1
        x += 1

def initFlames():
    global flames, moveFlamesFlag
    moveFlamesFlag = 4
    flames = []
    g = 0
    while g < 4:
        flames.append(Actor("flame"+str(g+1),(270+(g*20),
370)))
        flames[g].dir = randint(0, 3)
        flames[g].status = 0
        g += 1

def inputLock():
    global player
    player.inputActive = False

def inputUnLock():
    global player
    player.movex = player.movey = 0
    player.inputActive = True

init()
pgzrun.go()
```

Learn game development with Raspberry Pi

Get a head start with Raspberry Pi game creation with this resource list. By **Mark Vanstone**

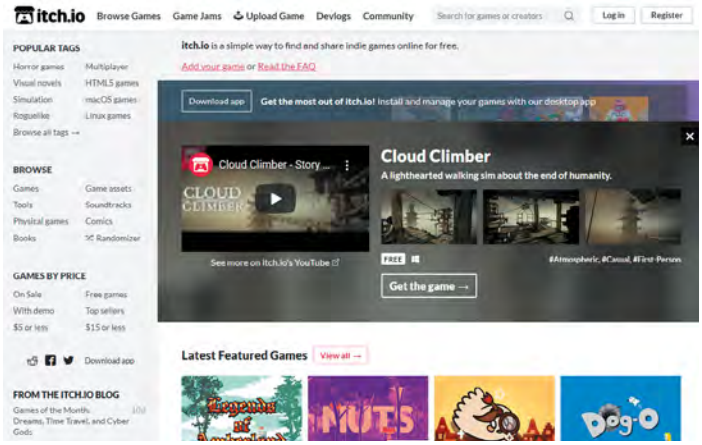
itch.io

CREATOR

Itch corp

Price:
Free / Percentage of sales

itch.io



For most developers of games, the main reason to create a game is to challenge others to play their game. So the first question is: how can we make a game available for other people to play? That's where itch.io comes in. The website provides an app store–style platform for independent developers to upload and sell their games.

Games can be built and uploaded in all kinds of formats. They can be built as executables, source code downloads, or online browser games. There is a large active community, and regular competitions to reward the best games. The itch.io site is free to use and provides lots of support for new developers and if you want to sell your game, they will deal with all the payment process but, of course, ask for a small cut of the profits.

Currently there are over 300,000 games hosted on itch.io, so you can have a good look around and see what everyone else has uploaded, and get ideas about how to present your new game to the world, get feedback from players, and even make a bit of money.

Books for game development

Paper-based or online books for reference and tutorial

ADVENTURES IN MINECRAFT

A treasure-trove of a book, both paper-based and for download. Learn to build games using the Minecraft engine, and even program external controllers to trigger events in-game.

magpi.cc/advminecraft

CODE THE CLASSICS

Learn how to create your own versions of retro games from scratch using Python and Pygame Zero. Five classic video games are remade, ranging from Pong to Sensible Soccer.

magpi.cc/codetheclassics

MAKE GAMES WITH PYTHON

An in-depth look at game creation with Python and Pygame. From your first game to physics simulations and alien invaders, this book is packed full of useful techniques and listings.

magpi.cc/makegamespython

Scratch

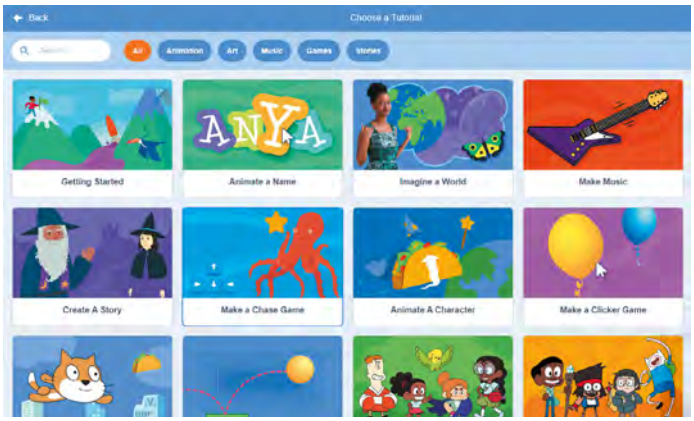
CREATOR

Scratch Foundation

Price:
Free

scratch.mit.edu

Scratch is available as a game and animation development system, both in a browser and as an offline program. Both work and look very similar. Scratch is an excellent introduction to programming, and provides a visual block interface to create interactive content. You can share games that are created with Scratch, and there are lots of examples on Raspberry Pi's website for you to see what others have done with Scratch. Graphics and sounds are included in the Scratch library, but you can also create your own using the built-in pixel editor or a separate paint package. There are extra extensions you can add to connect to external projects, and a whole range of tutorials to show you how to get started making the game of your choice.



Raspberry Pi Game Projects

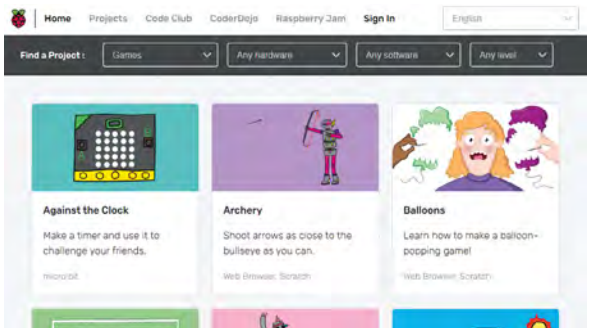
CREATOR

Raspberry Pi Foundation

Price:
From £19.99

magpi.cc/projects

Since the launch of Raspberry Pi, the Raspberry Pi Foundation has been producing example projects of all kinds on its website. In the game section, there are around 60 projects for you to delve into and find out how they were made and download the elements you need to build them. There are projects for Scratch, Python, web browsers, and even games to play with external hardware, like the Sense HAT. Each project describes what you will need to make it, shows the finished project, and then walks you through, step by step, what you need to do from beginning to end. You will also find suggestions for other projects to look at after you have finished, to progress further with your game development experience.



Game Creator Resources

Get free game resources online

PI GAME DEV

Pi Game Dev is a well-organised site dedicated solely to resources for making games on Raspberry Pi. There are comprehensive lists and links to game engines, art and music tools, code editors, and game assets.

pigame.dev

OPENGAMEART.ORG

OpenGameArt is a go-to, one-stop, free shop for 2D and 3D game graphics and sound effects. Searchable and categorised, this site features thousands of submissions from designers. You can submit your own creations to give something back.

opengameart.org

SPRITERS RESOURCE

Sprite sheets are bitmaps full of animation frames, and Spriters Resource has all the retro game graphics in sprite sheet format. Just search for your favourite game and there's likely to be a sprite sheet or two for it.

spriters-resource.com

ARCADE PROJECTS

INSPIRING PROJECTS AND STEP-BY-STEP BUILDS

> Build your own perfect and brand new arcade emulation machine with Raspberry Pi


START

MAKE YOUR OWN PINBALL MACHINE

This step-by-step guide breaks down the key stages of building the Princess Pinball table. While your table could be very different, the key components and techniques apply to a wide range of builds.

Every pinball table will need a shooter, flippers, bumpers, and rubbers. And tips – like using adjustable legs to help you get the perfect angle on your table – hold true across many different build styles.

Similarly, Martin Kauss's GPIO connection configuration and software to run your table's lights, sensors, sound, and scoring are powerful tools for any pinball build.

01 Set up the software

Start with a Raspberry Pi and a clean install of Raspberry Pi OS. You'll need an internet connection, and your life will be easier if you connect a monitor, which we'll also use later to track the player's score, keyboard, and mouse.

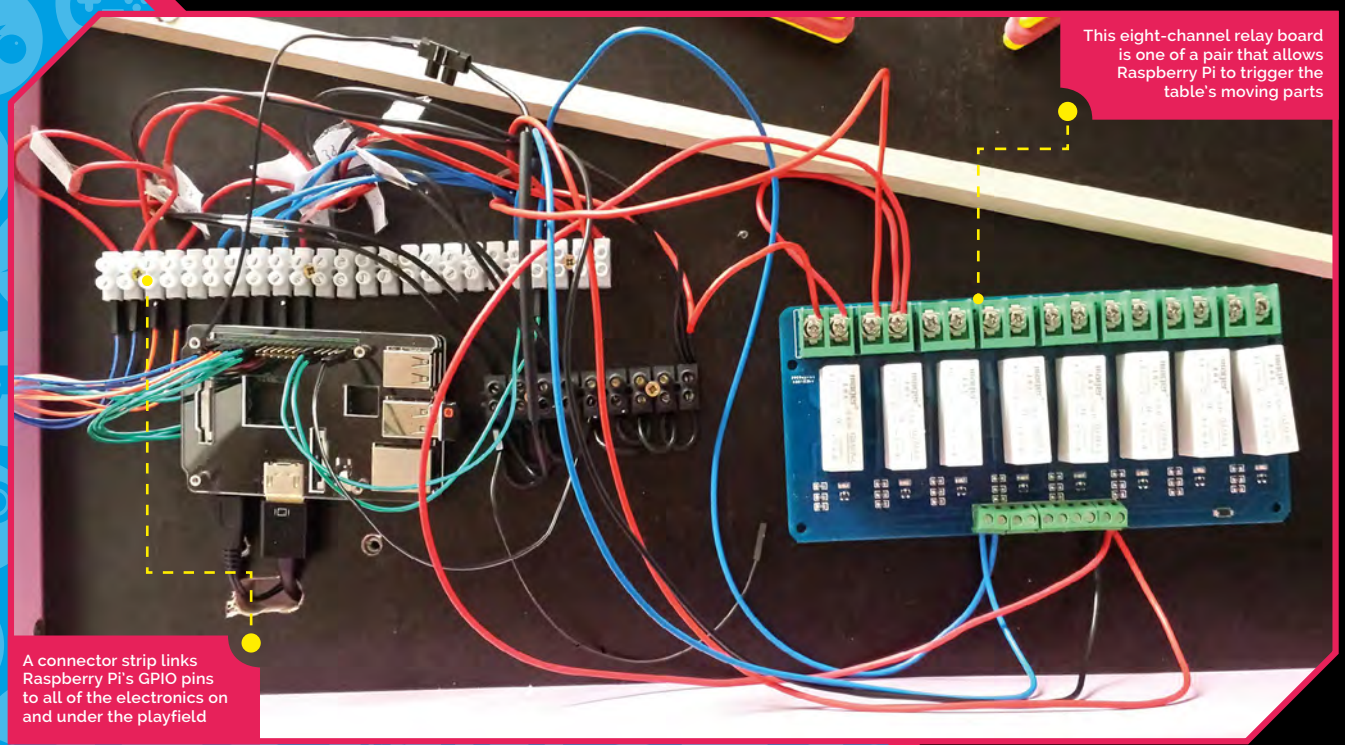
Open a Terminal window and type:

```
sudo apt install python-pygame
git clone https://github.com/bishoph/pinball.git
cd pinball
python pinball_machine.py
```

Pressing **Q** exits the program. Fonts aren't included, so to run the program you'll need to find your own **pinball.ttf** and **comicfx.ttf** TrueType files and copy them into **/usr/local/share/fonts/** – both are freeware and available online.

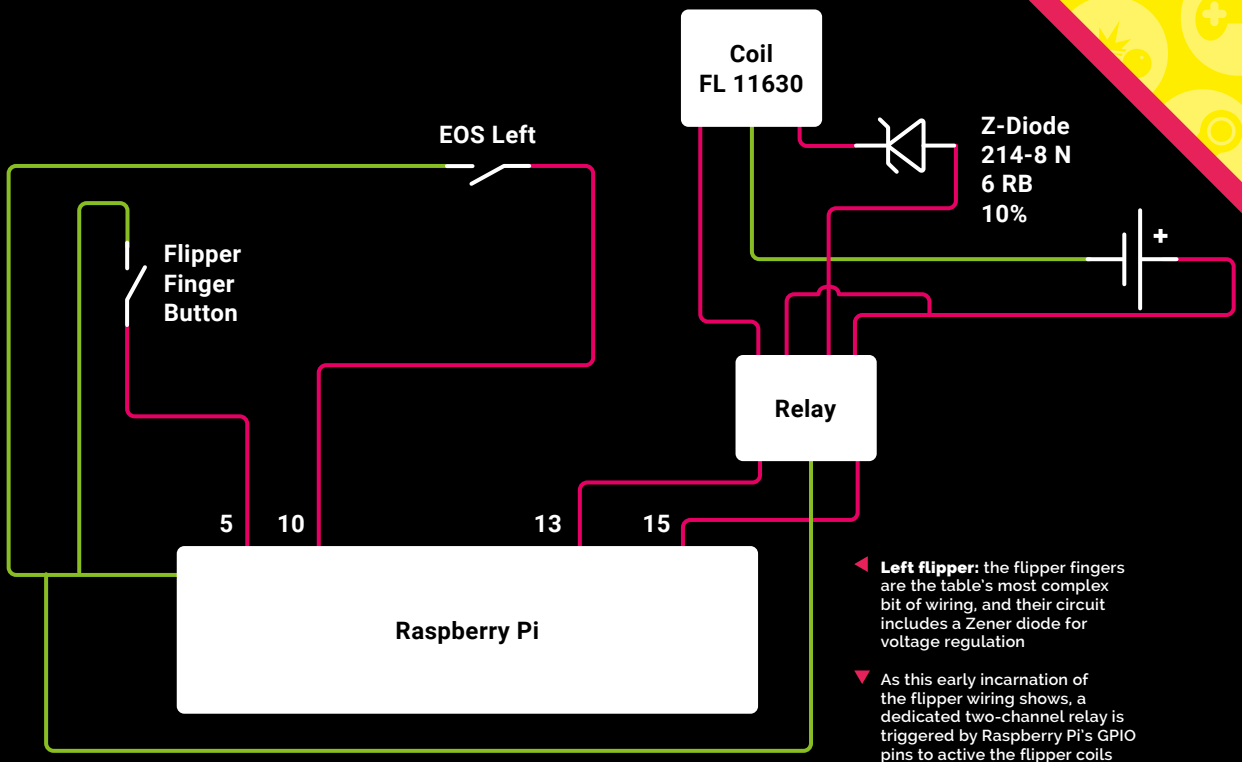
02 Vision on

Setting everything up is much easier if you've got a monitor connected to your Raspberry



This eight-channel relay board is one of a pair that allows Raspberry Pi to trigger the table's moving parts

A connector strip links Raspberry Pi's GPIO pins to all of the electronics on and under the playfield

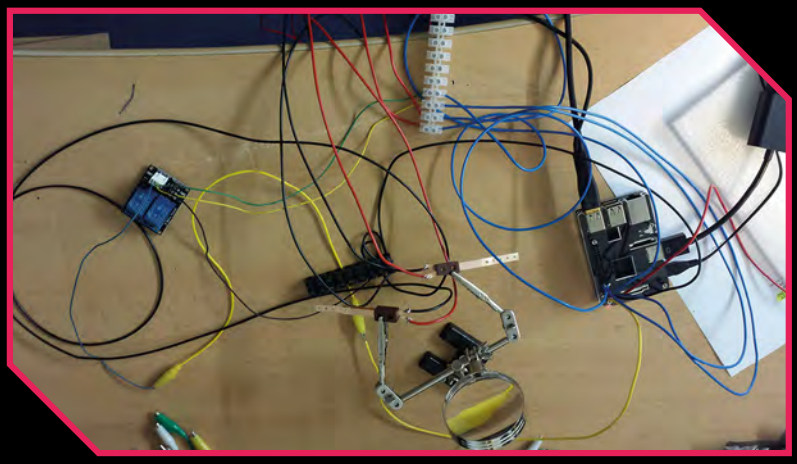


Left flipper: the flipper fingers are the table's most complex bit of wiring, and their circuit includes a Zener diode for voltage regulation

As this early incarnation of the flipper wiring shows, a dedicated two-channel relay is triggered by Raspberry Pi's GPIO pins to active the flipper coils

Pi, but our pinball table's Python scripts can also make use of an external display. We'll use this to show the player's current score, the number of balls left to play, the table's high score, and a few fun visual effects.

If you're feeling really sharp, you could use a VESA mount to affix the monitor to a backboard or stand attached to the table; or, if you're working with a bed frame like Martin, find space to attach it to the one-time head of the bed.



03 Face the music

You can play sounds through the integrated speaker of a monitor or by connecting speakers to Raspberry Pi's 3.5mm audio output. Sounds are defined in the **effects.py** script, which we installed in Step 01. You'll have to source your own audio files – Martin got some from **freesfx.co.uk**.

Place them in the **/home/pi/pinball/sounds** directory and edit **effects.py** accordingly. The script triggers sounds when the table is powered up, when your ball heads down the shooter alley, when it falls out of play by going down the outlane, and when it triggers the spinner or pop bumpers.

Samples in a bank called s2 are triggered as random events when the table is idle.

04 Frame and fortune

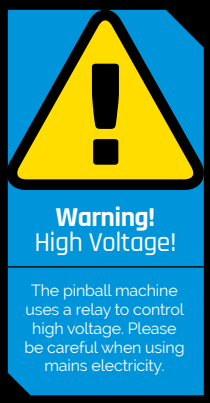
Martin Kauss's pinball table began life as a child's bed, decorated with colourful Disney princess imagery, but you could also build your own frame out of wood, buy a table base kit, or even make one out of K'nex or Meccano.

The frame measures 145×77cm and for the playfield – the surface of the pinball table, where all the action happens – Martin used a piece of 230mm thick multiplex board (plywood) with a black finish.

Beneath the playfield, you'll need space for your wiring and power supplies.

Test first

Make sure your components and connections work as intended before you permanently screw them into place.



Warning!
High Voltage!

The pinball machine uses a relay to control high voltage. Please be careful when using mains electricity.

05 More power, Igor!

This build calls for both a 5 V PSU, which handles the lights, and a 36 V one for the coils used to power the flippers and bumpers, which have comparatively high voltage requirements.

For example, although the pop bumper's coil is connected to the 36 V supply, its built-in LED light, like those elsewhere on the table, is powered by the 5 V PSU. For ease of connection, a GPIO stacking header is mounted on Raspberry Pi's GPIO.

From that, GPIO cables run to a connector strip. That includes both inputs and outputs, and all powered components such as lights and coils are wired up via relay boards. The relays are powered from Raspberry Pi (via physical pins 2 and 6) and the trigger action for each comes from the respective output GPIO pin.

It's useful to screw a small four-way plug bar to the rear or underside of your table, as you'll need separate power supplies for your monitor and Raspberry Pi.

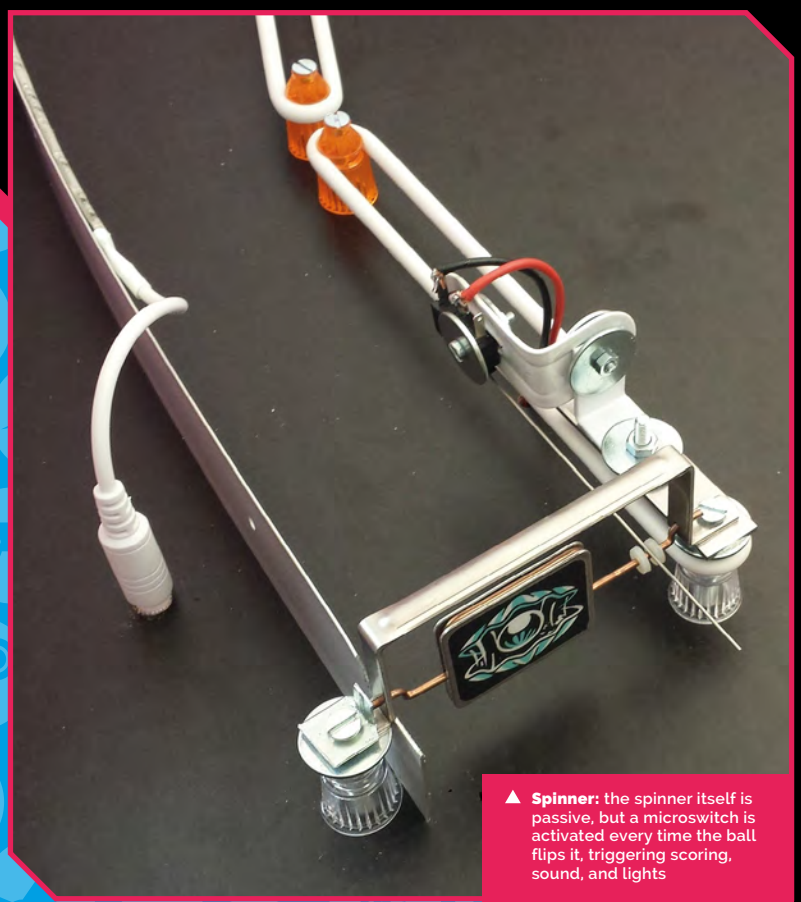
06 Tilt! Tilt! Tilt!

Pinball tables can't be flat, but getting the correct angle to ensure that the ball rolls towards the bottom at the correct speed can be tricky. Adjustable legs mean that you can easily set and change the table's incline.

In this case, four square wooden battens measuring 4.5×4.5×100 cm at the front and 4.5×4.5×110 cm for the rear have been used to make front and rear legs. The front pair have been made adjustable by using screw-in plastic feet which can be raised or lowered, allowing for a bit of trial-and-error engineering during construction.

07 Plan, sketch & drill

Once you've got your frame and a table surface to fit it, it's time to outline its features. Carefully measure, test-place, photograph, and draw around the flippers, shooter, slingshot, and



▲ **Spinner:** the spinner itself is passive, but a microswitch is activated every time the ball flips it, triggering scoring, sound, and lights

▲ You'll need a healthy selection of tools, electronic components, and other parts for a project like this, as well as the space to build it

bumpers you want to include. You'll also want to place side-mounted flipper buttons.

As well as these components, you'll need to use wood and metal strips for your lanes, an outlane area to direct the ball back to the shooter alley when it's lost, and the curved upper part of the table. Once you've marked up and double-checked, drill the holes you'll need to bolt on and wire up your components.

> Remember to leave enough space at the bottom of the table for your electronics

Remember to leave enough space at the bottom of the table for your electronics and Raspberry Pi!

08 Assemble the plunger

The plunger, also known as the ball shooter, is the first critical bit of your table to assemble. You can buy the whole assembly as a kit, including a rod, springs, housing, and mounting components such as the external trim plate.

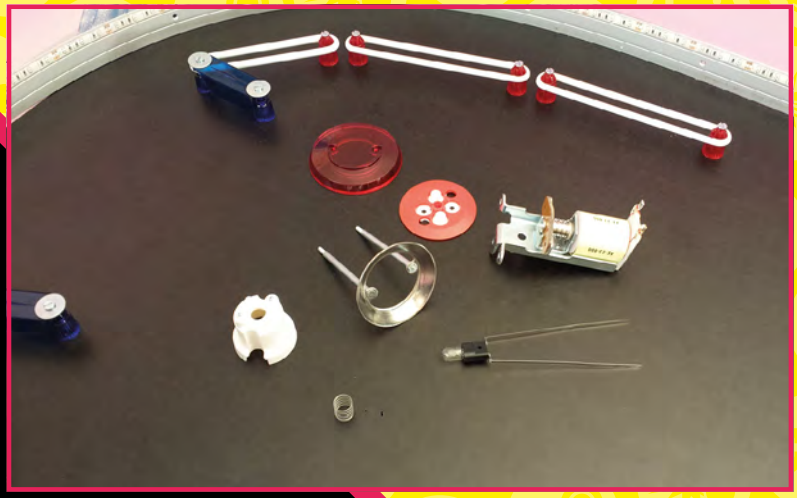
You'll want to put it at the left of your table, with the knob and external parts protruding from the front of your pinball machine. A length of wooden batten forms the lane that the ball will travel down, and the drain beneath the flippers should direct lost balls back to the shooter assembly. A high-tension spring holds a lane closure flap shut until it's forced open by the velocity of the ball coming through from the shooter.

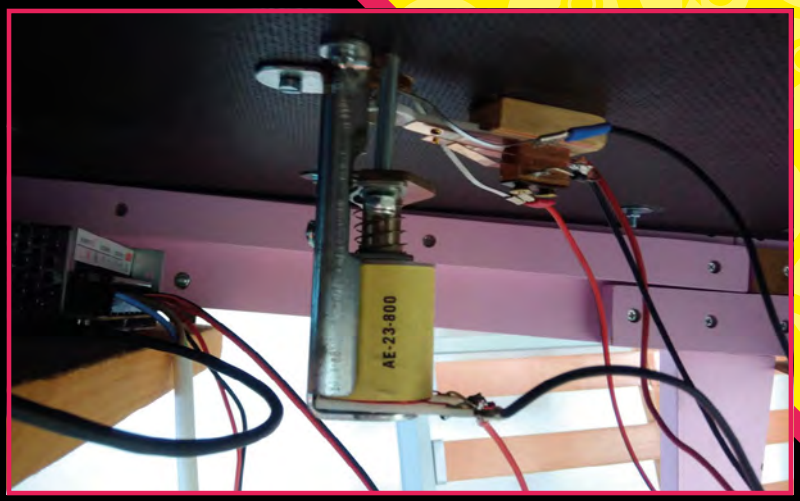
09 They see me rollin'

Most of the lanes and structural parts of this design are made from wood or aluminium strip bent into shape and held in place using wooden blocks screwed into the table, all of which makes for a pretty forgiving build. But you'll want your Raspberry Pi to be able to tell when the ball passes through those lanes.

For that, you'll need some rollover microswitches from a pinball part supplier, which you can fit to the table from below. Cut a channel in the playfield using either a router or a drill and jigsaw. Mount the switch on a strip of wood, and position it beneath the channel so that the switch protrudes enough to be triggered by the weight of the ball as it passes.

This build has rollover switch at the end of the shooter alley and on all the outlanes that take





▲ The pop bumper is assembled with its coil below the table and its skirt and cap above

Go online for more details

For extra information and a maker's blog about the Princess Pinball table, check out Martin Kauss's website at magpi.cc/iimYKq

the ball out of play, plus one at the very end, just before the ball re-enters the shooter alley, so the table knows when the ball leaves the playfield.

They're all connected to Raspberry Pi's GPIO header – via a connector strip – so they can trigger events such as lights sound and scoring.

10 Flip the finger

A pinball table's flippers are its most important components. You'll need a 36 V, 5 A PSU to give the flipper coils enough of a kick, and a two-channel relay to activate them when triggered by Raspberry Pi's GPIO.

Because we're using a Raspberry Pi to control everything, we'll need a modern flipper assembly with a normally open (NO) end-of-stroke (EOS) switch. The coils are mounted beneath the table's flipper fingers. For each flipper, you'll need to connect a button on the side of the cabinet to one GPIO pin, the EOS switch to another, and then another two pins – via the relay – to the flipper coil units, which actually have two different wire coils wrapped around them.

The first coil, HIGH, provides low resistance and makes the flipper finger move hard and fast, while the second, HOLD, has high resistance and allows you to hold down the flipper buttons to keep them upright.

Pin	Connector Strip Terminal	Type	Description	Relay Connection
3	1	IN	Flipper button right	
5	2	IN	Flipper button left	
8	3	IN	Flipper finger EOS right	
10	4	IN	Flipper finger EOS left	
7	5	IN	Spinner microswitch	
11	6	OUT	Flipper finger right HIGH	Relay #1,1
12	7	OUT	Flipper finger right HOLD	Relay #1,2
13	8	OUT	Flipper finger left HIGH	Relay #2,1
15	9	OUT	Flipper finger left HOLD	Relay #2,2
16	10	IN	Shooter alley microswitch	
18	11	IN		
19	12	IN		
21	13	IN		
22	14	IN	Bumper #1 switch	
23	15	IN	Bumper #2 switch	
2	16	IN	Slingshot switch	
26	17	OUT		
29	18	OUT		
31	19	OUT		
32	20	OUT	Light #1, shooter alley	Relay #3,1
33	21	OUT	Light #2, slingshots	Relay #3,2
35	22	OUT	Light #3, bumper	Relay #3,3
36	23	OUT	Light #4, bumper	Relay #3,4
37	24	IN	Outlane microswitches, one signal!	
38	25	IN	Bumper #1, coil	Relay #4,1
40	26	IN	Bumper #2, coil	Relay #4,2

The Princess Pinball table has proven to be a massive hit with Martin's kids and all their friends, complete with requests for a future multi-ball feature

11 Into the slingshot

Slingshot bumpers are the wedge-shaped components positioned just above your flippers, helping to bounce your ball back to them on its journey around the table.

They're made up of a set of star posts – brightly coloured plastic mounts that a rubber ring can fit around and a rubber surround for the ball to bounce off. A microswitch sensor detects when the ball hits, racking up points, and triggering noises and lights.

A pair of fancy slingshot covers rounds out the look, and a lane – made of wood – runs below them to provide a route to the flippers.

12 Add a bumper or two

Mushroom-shaped pop bumpers are among the most iconic elements of a pinball table. They're available in complete kits and most work by detecting, via an integrated microswitch, when the ball hits their plastic skirt. A coil then pulls down the bumper's rod and ring assembly to kick the ball away.

You can use the bumper's base to mark out where you want to mount it, with the coil mounted below the playfield surface. The coils are powered by the 36 V PSU and connected to Raspberry Pi via the eight-channel relay.

13 Add a spinner, connected via screw terminals

A spinner, or spinning target, is a classic pinball table component with its own lane and a microswitch that triggers lights and a score increment when the player hits it.

Here, the lane is created using a metal rail of aluminium threshold strip. The spinner is mounted using some DIY-shop metal brackets and bolts, plus star posts and rubber rings from a pinball supply firm.

A straight-wire actuator microswitch is connected to the spinner to detect the ball's passage and rack up points and flash lights around the table in response.

14 Lights! Action!

A pinball table is nothing without flashing lights. This project uses a 2 m LED light strip connected to the 5 V PSU to add some pizzazz to the table's shooter alley and orbit – the upper loop of aluminium strip that helps keep the ball rolling around the table.

There are also individual lights in each of two bumpers and the slingshot bumpers – a total of four GPIO connections on Raspberry Pi. The Python script we're running will activate the lights when the ball enters the playfield, hits bumpers, triggers the spinner, or passes over rollover switches. The lights are also triggered by the script at random events if there is no input or action on the table.

Fingers to yourself

Don't touch the coils or moving parts during testing, as the rapid contraction of the solenoid can deliver a painful pinch.

Part 01

Build an arcade machine: Get the parts

If you've ever wanted to build your own arcade machine, here's your guide. This month: the parts you'll need, how to choose them, and where to buy them

MAKER

K.G. Orphanides

K.G. is a writer, maker of odd games, and software preservation enthusiast. Their family fully supports the idea of an arcade machine in the living room.

@KGOrphanides

Over the following pages, we'll go through the process of sourcing, building, connecting, and installing a Raspberry Pi-based arcade cabinet.

While you can restore and convert a former JAMMA cabinet for use with Raspberry Pi, or build a cab entirely from scratch, we'll be taking the flat-pack route. This lets you build the cabinet of your dreams relatively easily, somewhat cheaply, and without recourse to full-on home woodworking.

This tutorial series will use an LCD screen due to the inconvenience of sourcing and potential issues with installing a CRT model, which carries the risk of a dangerous electric shock if not correctly discharged.

01 Choose your cabinet style

If you're after a classic upright one- or two-player cabinet, then you'll want either an all-in-one model or a 'bartop' cabinet with a pedestal or stand. Bartop cabinets can also be bought without the optional stand and placed on a table.

Flat 'cocktail' or 'coffee table' style cabinets are available in models for between one and four seated players and often use a vertically oriented screen, which can be split by software into two horizontal views for multiplayer games.

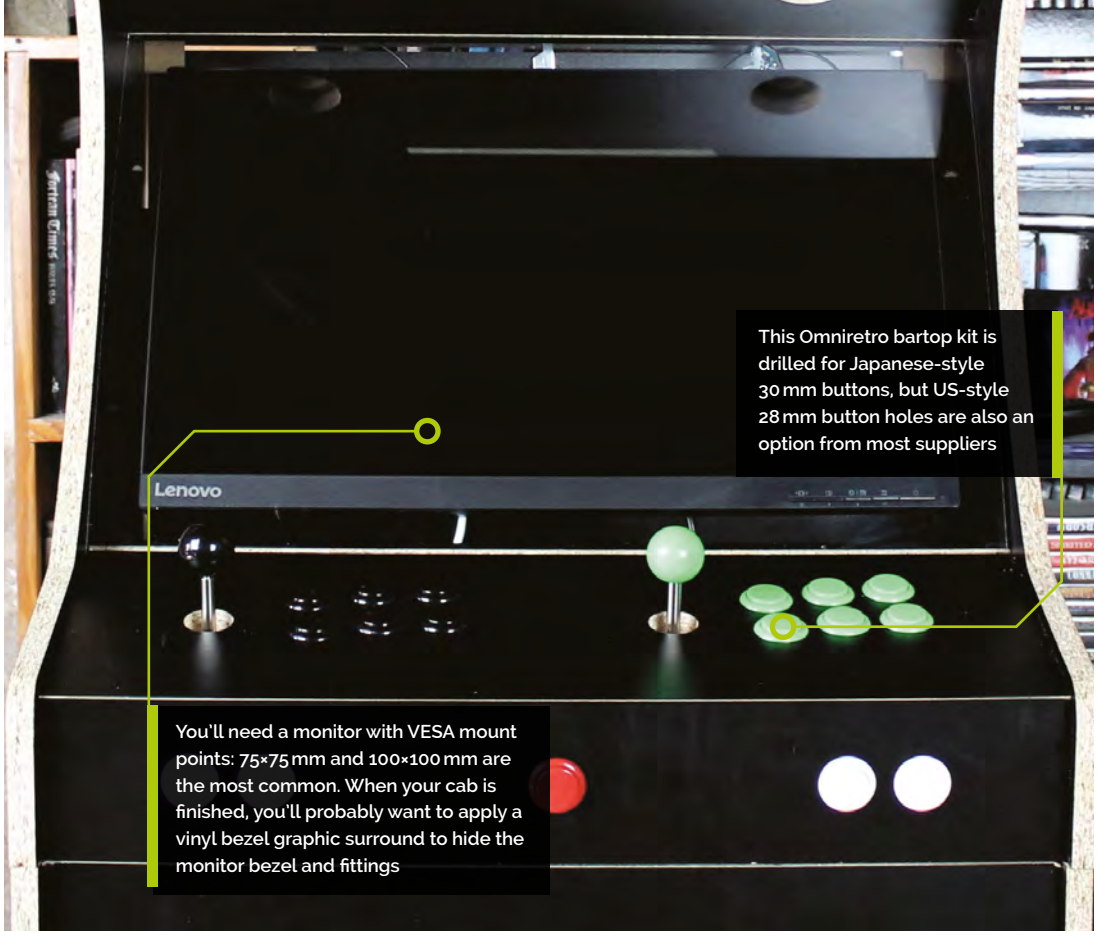
Other models include seated upright cabinets (often designed to take very large screens), angular tabletop models, and mini-bartops with 10-inch displays for those short on space.

02 Big screen glamour?

The size of your screen dictates the size of your cabinet, and vice versa. Before you start shopping, work out where you want the cabinet to live, and take height, width, and depth measurements.

If you're working with a 19-inch monitor, you'll likely get a bartop cab that's a little under 50 cm wide. This is the most practical choice if available space is limited. A 22-inch screen translates to a cabinet of a little under 60 cm, and a 24- or 25-inch screen means a cabinet width of a bit under 65 cm. You're generally fine fitting a smaller screen in a larger cabinet, but the end result won't look quite so polished.

Check the internal measurements of the cabinet against those of the monitor, including its bezel.



Top Tip

Button positioning

We're going with a six-button Japanese-style layout. Check out **magpi.cc/joysticklayout** to see some alternatives.

03 A good fit

Depending on the era of games you want to play, a large 1920×1080 widescreen display may not be the most authentic choice, but it is the most flexible, and modern emulators handle HD displays well.

Most cabinets have a VESA mount, usually in the form of a monitor support bar drilled for 75×75 and 100×100 mount points. Make sure your monitor has mounting points that match.

Finally, ensure that your monitor will work with Raspberry Pi: anything with a standard HDMI input should be fine, but older DVI and VGA displays require inconvenient adapter arrangements.

04 Materials

Self-assembly cabinets are usually made in MDF, but laminate, melamine, and veneer finishes are also widely available.

MDF swells badly if exposed to water, so if you're going to have drinks anywhere near your cabinet, a water-resistant finish is strongly recommended. If you buy an untreated MDF kit, apply and sand down between multiple coats of an MDF-specific solvent-based primer, then paint it to your heart's content, ideally with oil-based paint.

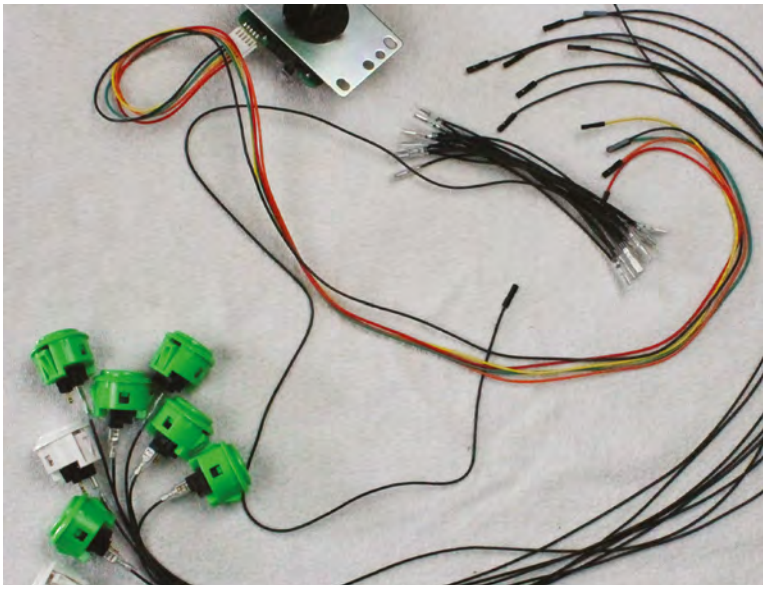
18 mm MDF is common, but you'll find cabinets in anything down to 10 mm for budget models. 18 mm or thicker construction materials may require a longer shaft or extender for your joystick. If in doubt, talk to the kit's supplier.

05 Finish and decoration

Regardless of the materials used, you'll probably want some plastic edging strip. This plastic trim helps to protect the edges of your cabinet, makes it easier to clean, and looks a lot more professional than exposed MDF edges.

Two types are popular. T-Molding is more secure but requires a slot to be cut for it to clip into – some DIY kits have ready-cut slots for this purpose, but budget models frequently do not.

U-Molding just clips over the edge. Cabinet makers will usually tell you how much moulding



You can get kits containing all the joysticks, buttons, and connectors you'll need; just make sure your button and cabinet hole sizes match

▲ A variety of compact bus- and mains-powered amp and speaker kits are available: this one takes power from the USB port and audio from the 3.5mm port

their kit will need and can usually supply the required quantity and type of edging.

Many arcade cabinet suppliers also sell a range of decorative and protective graphical vinyl sticker wraps. These should be applied with care to an appropriately finished surface (check with the sticker manufacturer for any finish requirements).

06 A giant screen protector

To protect your screen and create a flush finish, you can – and should – opt for an acrylic (polymethyl methacrylate, also known as plexiglass) screen protector. Again, this is something most self-assembly kits are designed to take and the majority of retailers will happily sell you one as either a standard part of the kit or an optional extra. Make sure you do opt in, as cutting your own plexiglass to precise dimensions can be a pain. Toughened glass and UV-resistant polycarbonate can also be used. You may need to add some standoffs to stop front monitor buttons being pressed by the screen protector.

07 The marquee club

Also included in kits as a matter of routine is a strip of acrylic for your cabinet's top marquee. You'll probably want to get a backlight-ready vinyl marquee (available from print shops, arcade suppliers, and on Etsy) to stick to this, but you could also decorate your own.

Sample shopping list

Here is an illustrative price list. The prices include VAT but not shipping or additional costs.

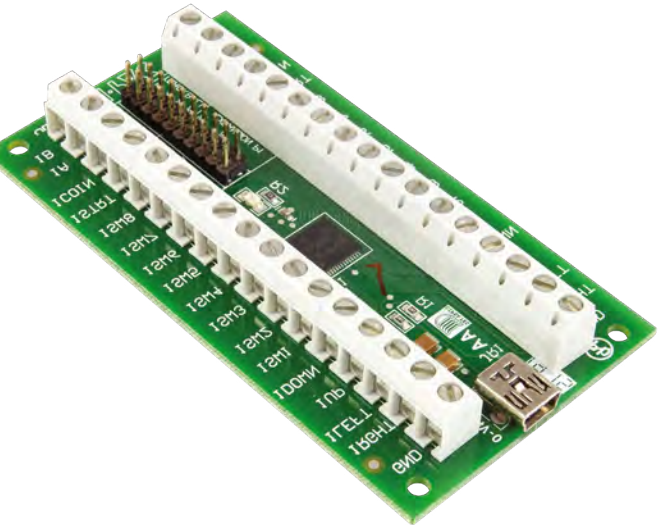
Item	Price
24-inch LCD monitor	£125.00
Bartop cabinet	£170.00
Bartop stand	£100.00
10 m T-Molding	£25.00
Acrylic control panel guard	£25.00
Two-player USB joystick + button kit	£70.00
Amp, speaker & cover kit	£25.00
Amp power supply	£12.00
Printed marquee	£6.00
LED strip lighting	£15.00
Molex power adapter for LEDs	£15.00
5-way plug bar	£15.00
TOTAL	**£603.00**

While you're at it, you may wish to get acrylic or metal panels to surround your buttons and joystick. These can be decorated, and protect your cabinet's surface, as well as providing a smoother feel. Button layouts tend to be standard, but these should ideally be bought from the same supplier as your kit for the best fit.

08 Raid the button tin

We'll be building a cabinet with an eight-way joystick and six 30 mm buttons, plus Start and Select buttons, for each player. A variety of alternative sizes and brands are available, with Sanwa perhaps being the most recognisable. You can order a cabinet with holes for extra side buttons if you're into digital pinball.

An easy cross-platform connection solution is a USB arcade encoder. Models by Zero Delay and Xin-Mo are popular, but the I-PAC 2 keyboard encoder has slightly lower latency.



If you want to use USB, the Ultimarc I-PAC 2 encoder is a popular choice that'll work with most computers. Check out **magpi.cc/ultimarcgit** for advanced configuration

09 Pick a driver

You can connect controls to Raspberry Pi's GPIO, using either the Adafruit Retrogame (**magpi.cc/adaretrogame**) or mk_arcade_joystick_rpi (**magpi.cc/mkjoystick**) drivers – we'll be using the latter.

Arcade joysticks generally use a five-pin JST connector, while non-illuminated buttons each have a pair of quick-connect spade connector fittings, one of which must go to ground. Spade to DuPont GPIO cables are uncommon, but can be bought either individually or as part of a kit from specialist retailers such as SmallCab. Illuminated button kits are available with an extra external PSU.

> LED strip lighting is a popular choice for marquee panels

10 The sound of success

It's a good idea to order your cabinet with a couple of pre-drilled speaker holes and covers to go over them. The most common option for audio is an externally powered stereo amp, connected to Raspberry Pi's 3.5mm port, and 10cm/4-inch speakers, but USB-powered kits are also available. If you have one lying around, you could also consider mounting a compact USB sound bar behind your speaker grilles.

11 More power, Igor!

A major advantage of this kind of arcade machine build is that there are no internal power supplies to bother with. There's enough space to mount a plug bar inside most cabinets, and you can use this to power the monitor, Raspberry Pi, and any extra transformers required for lights or speakers.

Where to buy

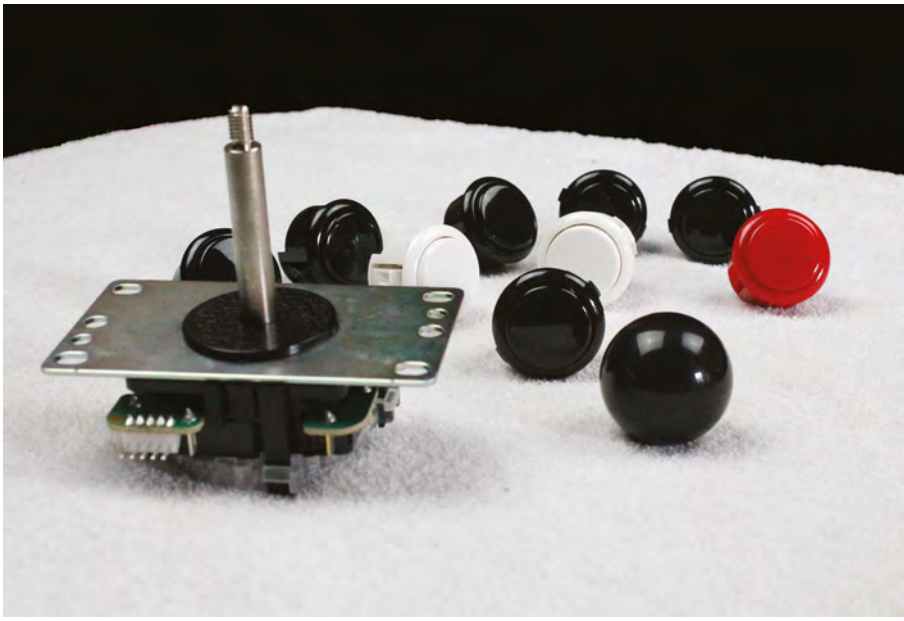
There are a number of UK and EU retailers specialising in self-assembly arcade cabinets and components. While it's easiest to get everything in one place, you have to mix and match for specialist components such as GPIO-compatible wiring looms.

- **Arcade World UK** – arcadeworlduk.com – supplies a wide range of kits and components; discount codes available for most non-furniture items
- **Bitcade** – magpi.cc/bitcadekits – UK arcade machine maker that also supplies kits
- **Omnireto** – omniretro.com – Spanish firm with a notable budget range
- **Rockstar Print** – rockstarprint.co.uk – custom marquee and wrap printer
- **SmallCab** – smallcab.net – French supplier of arcade kits and hardware including GPIO-friendly wiring

LED strip lighting is a popular choice for marquee panels, but you'll need to buy a Molex power adapter to go with it, or repurpose a PC power supply. You can run a plug lead out of the back or optionally install an external power socket and switch, if you're comfortable with simple electrical wiring.

12 Room to build

Before you start ordering, consider not only the space you have to house your cabinet, but also how much room you have to build in. Don't get an untreated MDF cabinet unless you have a large, ventilated (and paint-resistant!) space where you can apply primer to each part, as well as appropriate eye and breathing protection.



You'll want to source durable joysticks and buttons for your arcade machine

Warning!
Paint and dust

When sanding, sawing, or painting, be sure to use appropriate eye and breathing protection in a well-ventilated space.

magpi.cc/diysafety

Part 02

Build an arcade machine: Assemble your cabinet

Once your arcade cabinet kit arrives, it's time to put everything together

In this tutorial, we will assemble an arcade cabinet, fit controls, and mount a monitor. You should follow the video or written assembly instructions for the model you buy, but we'll go through the process so you know what to expect and how to handle the awkward bits.

Kits don't necessarily come with the screws and bolts you'll need to attach parts such as speakers, speaker grilles, and monitors, so check that you have all the hardware you'll need before you start.

Our cabinet is an Omniretro Bartop Arcade King with a stand (**magpi.cc/kingbartop**), made of 16 mm black melamine laminate, and we are using a 24-inch monitor.

01 Lay out your parts

MDF and melamine laminate are light, cheap, and sturdy when assembled, but can be susceptible to damage if dropped or pivoted hard on an edge or corner.

Make some space and put down towels to protect the cabinet parts and your floor from one another. If your unit consists of a separate bartop and stand, build them one at a time. Read or watch the manufacturer's instructions and make sure that you have all parts, fixings, and tools to hand before you start.

02 Preparation

Assembly varies from brand to brand. If access to the assembled cabinet is restrictive, you may have to fit your buttons and joystick to it before you put it together.

Similarly, attach speakers to the inside of the marquee bottom and speaker grilles to the outside before you assemble the cabinet. If you're working with laminate, mark up the screw positions through their holes with a paint pen and use a 3 mm bit to drill pilot holes.

If you've already decided on your marquee, control panel and bezel graphics, your life will be easier if you apply these to their acrylic sheets before assembly (we'll be looking at this in detail in a later tutorial).

> "Put down towels to protect the cabinet parts and your floor from one another"

03 Assembly

If you're comfortable with self-assembly furniture, an arcade cabinet shouldn't present too much trouble, but a second person can be helpful for fitting and moving awkward parts.

Ours has a control panel with a hinged access door beneath it, so we attached this hinge first

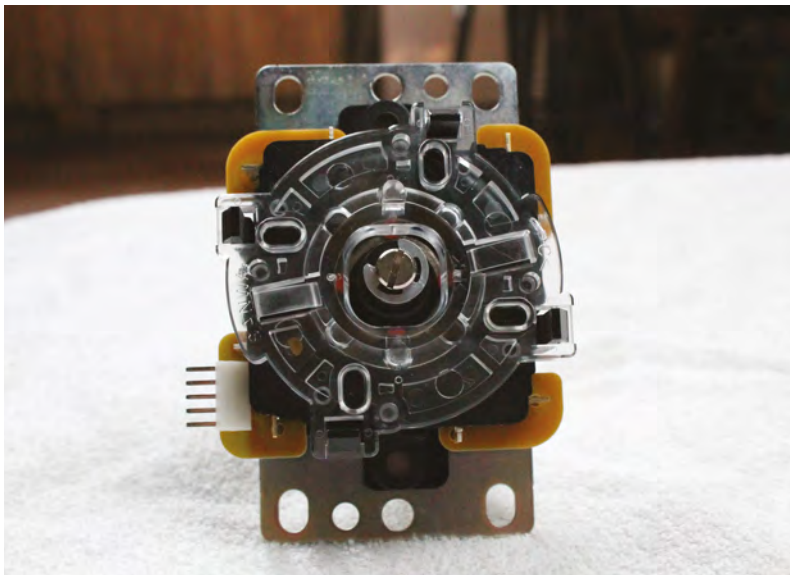
Top Tip

Snap-out

Snap-in buttons can be hard to remove without damage. The ButterCade Snap-out tool for push-buttons (**magpi.cc/snapout**) is a plastic device to help with this.

You'll Need

- Screwdrivers, spanners, Allen keys, crimping tool
- Cordless drill
- Drill bit set. Screwdriver bits, drill bits, countersinks, tank cutters
- Additional bolts, screws, female spade connectors (to mount components)
- Dremel (recommended) and 3 mm drill
- Paint pen (silver if you have black laminate, black for MDF)
- Old towels or sheets to protect parts
- Foam cleanser and microfibre cloths (to clean your cabinet and acrylics)



▲ The underside of a Sanwa JLF-TP-8YT joystick. Note the e-clip securing the central shaft

then set the panel aside. We then attached the hinge for the bartop's rear access door and base, lined this part up with the cabinet's hood-like top, and bolted all of these parts to one side panel laid on top of them.

Lining bolts up with pre-drilled holes for this kind of build can be fiddly. If you have trouble, screw the bolts through the side panel until they're protruding, and use them to help find the correct positions.

04 The control panel

With one side now in place, slide in the control panel and bolt it to the same side as the other parts. Next, attach the marquee bottom that houses the speakers, which should already be mounted at this point.

With this model, we then close the latch on the rear access door and carefully flip the entire cabinet over onto the now-secure side panel. This is the best time to slide the marquee and screen acrylic panels into place. If you've not already applied graphics to them, leave their protective film on – it's easy to peel off later.

We now position the second side panel. We recommend again screwing in the bolts until they just protrude from the opposite side to help you lower the panel securely and accurately onto its pre-drilled holes.

05 Feel the power

Drill a hole at the back of your bartop and run the power bar's cable out through it to connect directly to a plug socket.

Some suppliers will wire a socket and bar for you, but note that international plug standards differ. Use a plug bar that can be surface-mounted inside of the cabinet.

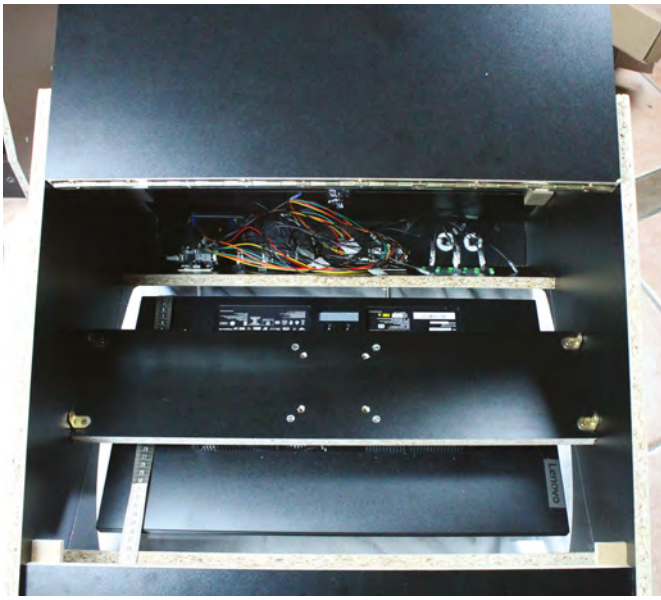
While you're back there, cut a hole to accommodate a booted Ethernet cable or, more tidily, a screw-down Ethernet extension port. This will make Steam Link game streaming easier.

06 Extending your shaft

If your cabinet is over 16 mm thick, you'll want a longer than standard joystick shaft. Shafts are easy to swap, but watch out for parts dropping out.

Like most sticks, our Sanwa JLF-TP-8YT's shaft is held in place at the bottom by an e-clip. Hold the unit upside down, press on the bottom of the shaft with your thumb, and use a small flat-head screwdriver in your other hand to pull the clip off, using the slots in it. Pull the old shaft gently out from the top and push the new one in, carefully setting the pivot at the top and the spring and black plastic actuator at the bottom into place.

Use a thumbnail to depress the actuator and slide the e-clip back into place. You can also use pliers or your screwdriver to help push it on. For a demonstration, see this YouTube video on changing joystick shafts: **magpi.cc/joystickshafts**.



▲ To fit the VESA mount, place the cabinet face-down, then put the mounted monitor face-down on the front acrylic screen. Use a tape measure to help with positioning

Warning!
Mains electricity & power tools

Be careful when handling projects with mains electricity. Insulate your cables and disconnect power before touching them. Also, be careful when using power tools during this build.

magpi.cc/drillsafety
magpi.cc/electricalsafety

Cable tidy

Cable lacing is a cable management technique where a nylon cord is used to bind wires together. It can be used to create incredibly neat builds, like this Arcade Stick by Gordon Hollingworth, Raspberry Pi's Chief Product Officer.

Gordon learnt to cable-tidy this way as an apprentice for the MOD. "Tying the knot has to be done in a very specific way to avoid it looking untidy," he tells us, "basically a capital offence in the apprenticeship!" Gordon's cables have knots regularly at 1cm, which keeps them smart. "We learnt this way because when you put a box into a plane or tank with some equipment in it, the vibration will shake apart pretty much any connection in the first hour. So this was the way it was done when electronics was more about wires connecting things than PCBs."

You can buy nylon cord and learn more from RS Components (**magpi.cc/cablelacing**).

▲ Snap-in buttons are held in place by plastic clips. Connect your DuPont GPIO cables first to make internal wiring easier

> "There's room to slide the screw slots on most joystick mounting plates"

07 Installing your joystick

Two plastic dust washers come with Sanwa joysticks. Slide one onto the shaft before you mount the stick onto the underside of your control panel.

When mounting your joystick, position it, mark up the position of the top right screw-hole on the joystick's baseplate with a paint pen, and drill a pilot hole, being careful not to go all the way through.

Attach your joystick by that screw, make sure it's centred, and mark up the next hole or holes. There's room to slide the screw slots on most joystick mounting plates, so you've got a bit of wiggle room when it comes to the final fit.

Don't worry too much about the orientation of your joystick – position it where it won't get in the way of the rest of your wiring. These are nominally designated up, down, left, and right positions; you can reassign these through wiring and in software.

Finally, slide the second dust washer onto the shaft on the other side and screw the joystick's ball on.

08 World of buttons

Snap-in buttons are ideal for thick wooden cabinets – plastic clips hold them in position inside the holes drilled for them. If you have an acrylic cover for your control panel, the buttons will hold it in place.

It's a good idea to attach your spade connectors to DuPont GPIO jumper cables before installing them, but you'll have to connect the shared ground cable after they're in place. We wired GPIO to the right and shared ground to the left connector on each button, but it doesn't matter which goes where.

Where you have longer stretches between buttons, skip a connector on the ground chain to give yourself some extra cable to play with.

You can label the end of each GPIO cable for later ease of connection to Raspberry Pi, but they're not too hard to trace in most cabinets.

Top Tip

Foot the bill

To help the cabinet stand on an uneven floor, you can fit four rubber feet to its underside.

▲ To make it easier to line up the sides of your cabinet with their pre-drilled recieving holes, partially screw in each bolt until a couple of millimetres protrude on the far side

> "If you find that you now can't reach or fit a part, don't panic"

▼ Before construction, lay out the parts of your bartop on some old towels to protect them

09 Bolt screen to VESA mount

Screen mounting can be fiddly. Most cabinets come with a baton-like wooden VESA mount that's designed to be screwed into place from the inside. Start by bolting your monitor to the mount. Unless you're working with a specialist cab designed for giant screens, you'll be using a 75 mm or 100 mm VESA mount. These usually take M4 bolts and have a depth of 10 mm. So if bolts aren't included, you'll need four, at a depth of 10 mm plus the depth of your mount, although you can get away with shorter if you countersink them.

10 Screw VESA mount to the cabinet

Place the bartop face-down on the ground, protecting it with a towel. Take the protective plastic off the acrylic on the inside of the cabinet. Clean the acrylic with a microfibre cloth and anti-static foam cleanser

Lay the monitor, attached to the cabinet's VESA mount, face-down on the acrylic inside your cabinet. On the interior sides of the cabinet, mark up the position of the holes in the brackets on each edge of the VESA mount. Remove the mount, drill pilot holes, then replace and screw down the display and its mount.

If your monitor has a front power button, you can use adhesive chair leg floor protectors as soft spacers to stop it from being pressed by the acrylic screen.

11 Don't panic

If you miss a stage in your build and find that you now can't reach or fit a part, don't panic. Speakers – and any other components in need of securing – can be attached internally using strong double-sided foam tape.

Most external parts can be drilled and fitted in situ. If you want to deal with decoration last, then you can sometimes pop out your acrylic panels or, better, remove one side and reattach it.

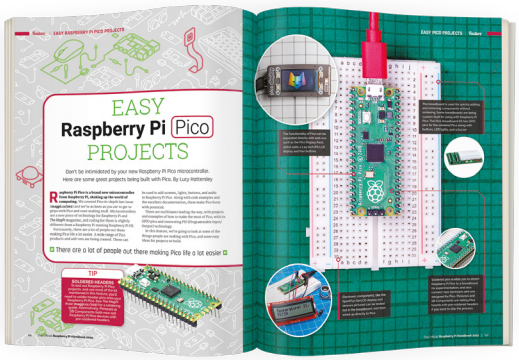
As you'll see from the photos, we have temporarily applied U-moulding to protect the edges of the cabinet. U-moulding is easy to remove and refit or replace, assuming you don't glue it down, but T-moulding is a little harder to remove cleanly.

We're now ready to connect Raspberry Pi. That will be covered in the next tutorial.

THE *Official* RASPBERRY PI HANDBOOK 2022

200 PAGES OF RASPBERRY PI

- QuickStart guide to setting up your Raspberry Pi computer
- Updated with Raspberry Pi Pico and all the latest kit
- The very best projects built by your Raspberry Pi community
- Discover incredible kit and tutorials for your projects



Buy online: magpi.cc/store

Part 03

Build an arcade machine: Command and control

We've assembled our cabinet. Now it's time to put Raspberry Pi to work with the Recalbox arcade OS

With our arcade cabinet built, it's finally time to get emulating with Raspberry Pi. We're using the Recalbox emulation distro for this project, which has excellent GPIO arcade controller support, a slick front end, and a handy web interface to help you configure and manage it.

The RetroPie distro is a popular choice for arcade machines, and adds Steam Link support, but requires manual installation and pull-up switch reconfiguration to get GPIO arcade controls working.

01 Wire up your controls

Last month, when we added buttons to our cabinet, we recommended attaching the spade-to-DuPont cables that will connect to Raspberry Pi's GPIO before inserting the buttons. If you didn't, it's time to open the back of your cab, grab a torch, and get in there to fit them.

Connect a spade-to-DuPont cable to each button and connect a shared ground cable to each of the left and right button banks. Where you have longer stretches of buttons – for example between the central hot button connected to player one's rig and the player one start button – it's a good idea to skip a connector on the ground chain to give yourself some extra cable to play with.

Plug the 5-pin cable into the joystick. Looking at our Sanwa stick from below, the bottom-most pin, which connects to the black cable strand on standard-coloured 5-pin wiring harness, is ground.

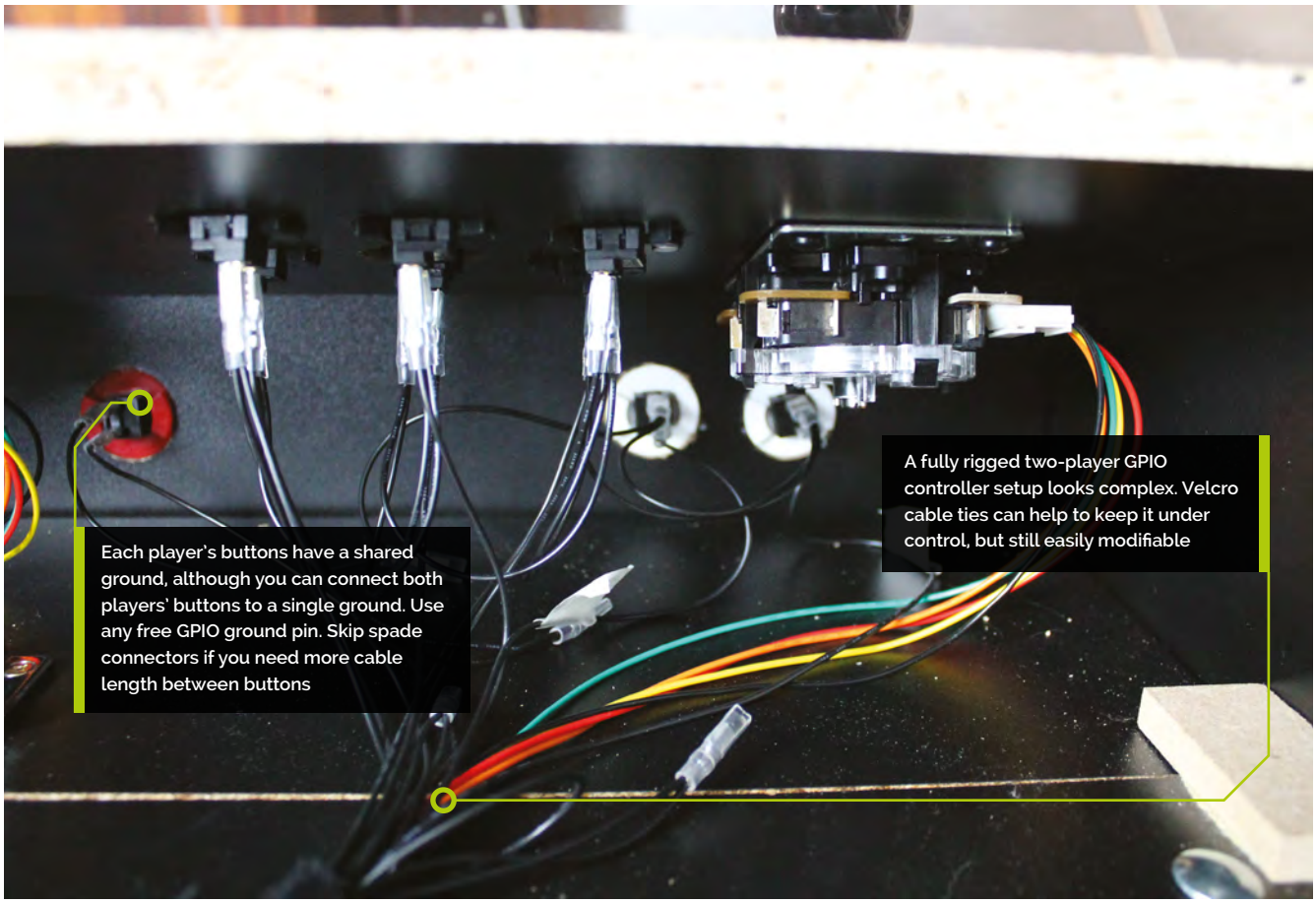
▼ Recalbox's main menu takes a couple of button presses to get to but has a comprehensive set of configuration options

02 Connect to GPIO

This is the fiddly bit. We suggest using a case for Raspberry Pi that fully exposes the GPIO pins. The GPIO wiring diagram shows which buttons, directional controls, and ground connections should be attached to each pin. While buttons and controls can be reconfigured in software, ground cannot. Our setup uses a total of 25 GPIO inputs, plus four ground connections. Input 25 is for a dedicated hotkey button.

03 Install and power up

Open Raspberry Pi Imager, connect your microSD card writer, and Choose OS > Emulation and game OS > Recalbox and the version of Recalbox that matches your Raspberry



Pi. Click Write and wait for the image file to be written to the microSD card. When Imager has finished, remove the microSD card and insert it into the Raspberry Pi in your arcade build. Connect the cabinet's monitor and speakers to Raspberry Pi. Plug in a keyboard on a long cable. Plug in Raspberry Pi's power and it will boot to Recalbox's EmulationStation interface, which you can immediately navigate using the keyboard. However, we still have to enable our GPIO arcade controls, wireless networking, and other configuration options.

04 Connecting Recalbox

Recalbox has SSH and Samba enabled by default, as well as a web interface available via your browser on **recalbox.local**. Recalbox should appear on your network as RECALBOX (File Sharing).

> Recalbox has SSH and Samba enabled by default, as well as a web interface

A wired Ethernet connection will give you immediate access to these. If you don't have one, press **ENTER** to open the menu, scroll to Main Menu, and select it with **A** on the keyboard, then select Network Settings, enable WiFi, select your SSID, and then select 'Wifi Key' to enter your password. Recalbox only has a root user. The default username is **root** and the password is **recalboxroot**.

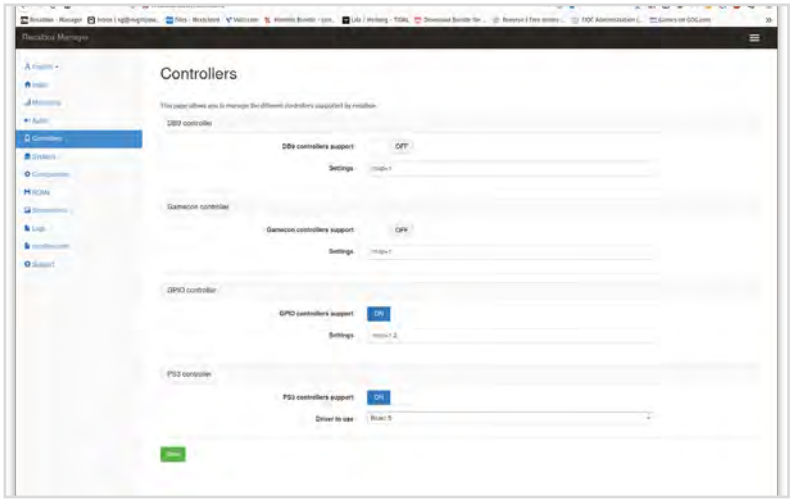
05 Configure Recalbox

You can access Recalbox's config file – **recalbox.conf** – by connecting via SSH, by browsing to the system directory in the Recalbox (File Sharing) Samba share, by pressing **F4** and then **ALT+F2** at the cabinet to exit to the console, or by going to **http://recalbox.local/** and selecting the **recalbox.conf** tab in the left-hand menu pane.

Under 'A – System options, Arcade metasystem', remove the semicolon that comments out `emulationstation.arcade=1`. This will make the arcade category the first entry in Recalbox's EmulationStation interface.

You'll Need

- Spade to DuPont cables
- Spade to DuPont shared ground cables
- Joystick to DuPont cables
- At least one Neo Geo Classics Collection game **magpi.cc/ironclad**



A web interface at http://recalbox.local gives you control over almost every aspect of your arcade machine's setup

Under D2 – GPIO controllers, set `controllers.gpio.enabled=1`. Save your changes and, at the arcade cabinet, open the menu, go to Quit > Fast restart Recalbox.

Top Tip

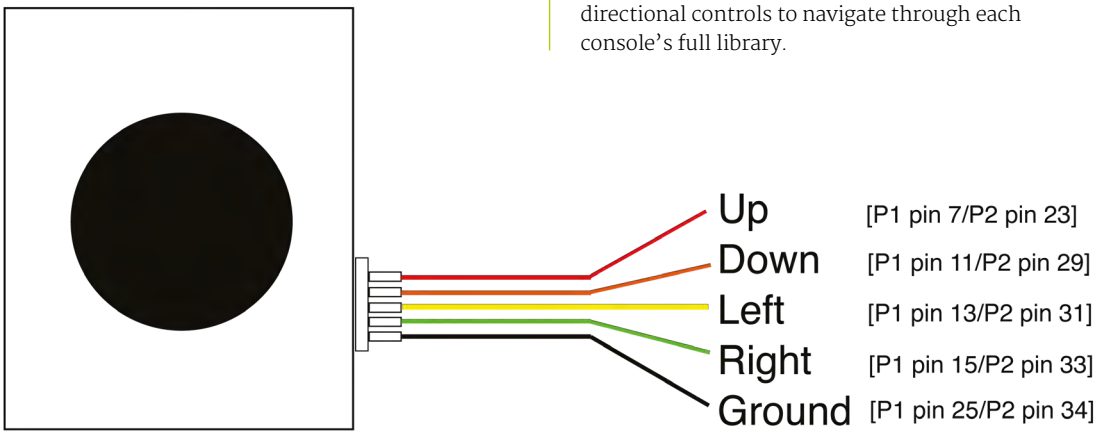
USB controls

To convert your controls to USB, use a Xinmotek board (**magpi.cc/xinmotec**) instead of connecting to GPIO.

06 Optional: Take control

Recalbox will now automatically detect GPIO controllers and, if all your buttons are wired as it expects, will already have the correct button configurations. Use the bottom-left button (B) to select options and the bottom-centre button (A) to go back. Left and right navigate between systems; up and down navigate between games or options within a menu. Press Start to open the configuration menu.

If your buttons aren't connected in that order, or if you prefer an alternative layout, open the menu and go to main menu > controllers settings > configure a controller. Press down to skip an entry that you don't have buttons for. If you don't have a hotkey button for one or more players, set it to Select.



Viewed from below, a standard Sanwa joystick's 5-pin connector goes to up, down, left, right, and ground. The diagram shows standard colour coding

07 Sounds good

If you have no sound, open the menu, select sound settings, and check the output device. We had to switch to 'headphones – analog' output to use our cabinet's speakers, connected to the 3.5 mm output on Raspberry Pi.

Recalbox plays background music all the time by default. This is charming, but a bit much for a cabinet that lives in the sitting room. Switch the Audio Mode to 'Video sound only' – to only hear the splash screens on boot – or 'No sound'.

If you prefer, you can add your own music by copying it to Recalbox's **share/music** directory.

08 Getting to know Recalbox

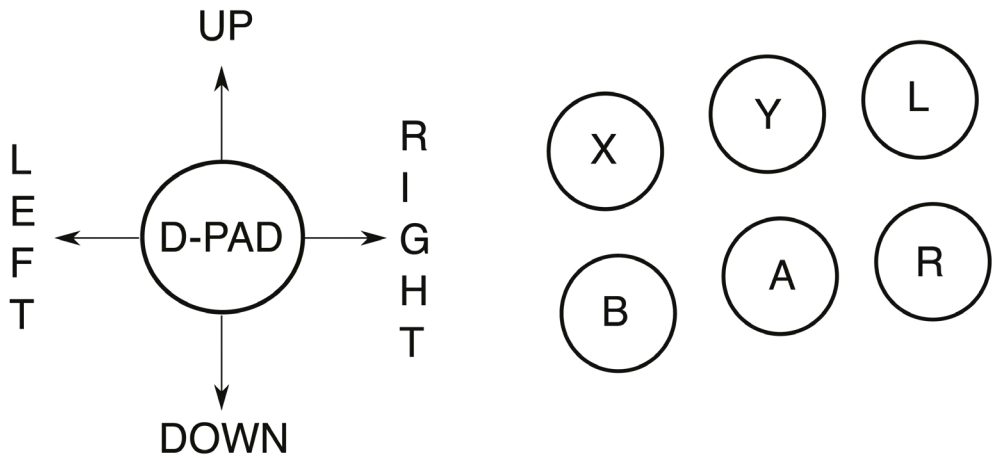
Recalbox comes preloaded with a number of freeware and open-source games. Because we enabled arcade mode, this category appears first. There are already four games loaded into it.

Select the category by pressing button B and scroll through them with the joystick. Gridlee, released in 1982, looks great for the era. Press B to load it.

Press Select to add credits and press Start when you're ready to play. When you've had enough, press the hotkey button and Start together to quit back to the Arcade menu.

> "Press the hotkey button and Start together to quit back to the Arcade menu"

You can press A to go back to the top menu, and use the joystick to navigate up and down through the list. But it's easier to use the right and left directional controls to navigate through each console's full library.



Button and joystick correspondences for player controls. The joystick maps to the D-pad. L and R correspond to L1 and R1, equivalent to the shoulder buttons of modern joypads

09 recalbox.local

Once your arcade machine is connected to your local network, you'll be able to access it in a web browser via **http://recalbox.local**. On the main page, you'll see shortcuts to a virtual gamepad, keyboard, and touchpad, which allow you to navigate through the arcade machine's menus remotely.

To add some authenticity to older titles, go to Systems and set the Shader set to Retro, which will apply community-favourite shader and scanlines settings to each game. On the other hand, if performance is poor, disable shaders and rewinding here. Click Save at the bottom of the page to store your changes.

Below, the Configuration tab lets you set networking options, enable and disable the Kodi media player, and configure the behaviour of the EmulationStation front end and the hotkey.

10 Manage game & BIOS files

The easiest way to manage your game ROMs on Recalbox is via the web interface, where the ROMs tab lets you select the directory for your desired console, stop the EmulationStation front end, upload games, and restart EmulationStation to load them.

You can also copy games over to the **roms** directory in Recalbox's Samba share. Even if you don't plan on emulating a specific console, don't delete the containing folders for its games, as they're required.

Recalbox also shares a **bios** directory, where you can add freeware or legally purchased computer or console BIOS files.

11 Buy and install a game

ROMs and a functional BIOS set for a number of Neo Geo maker SNK Corporation's games are available to buy as part of the Neo Geo Classics Collection (**magpi.cc/neogeoclassics**).

You'll need a Windows, macOS, or Linux PC to install or extract these. You'll find the ROM and BIOS files in the install directory; for example, **ironclad.zip** and **neogeo.zip** respectively for the fantastic scrolling shoot-'em-up Ironclad. If you don't want the whole collection, you can buy Ironclad alone at **magpi.cc/ironclad**.

Connect to Recalbox via SMB and copy the game ROMs into **roms**, and **neogeo.zip** into **bios**.

Restart EmulationStation and you should find your new games in the Arcade game list. Not all of them will work out of the box. Start any of them and press the hotkey and B buttons to open the Libretro emulation interface. Scroll down and select Options > Neo-Geo mode > Use UNIBIOS Bios. We aren't using UniBios here, but the file supplied by SNK is compatible with this setting.

Warning!
Mains electricity & power tools

Be careful when handling projects with mains electricity. Insulate your cables and disconnect power before touching them. Also, be careful when using power tools during this build.

magpi.cc/drillsafety
magpi.cc/electricalsafety

Top Tip

Preconfigured USB support

If your cabinet uses a USB controller board, then RetroPie won't need any extra drivers to detect your controls.

▲ SNK has made plenty of its arcade ROMs available to buy. Ironclad for Neo Geo-based arcade machines is a particular favourite

▶ GPIO wiring: Connect your joysticks and buttons to Raspberry Pi's GPIO as shown. *Image by digitalLumberjack of the Recalbox project, licensed under GPL2*

Press A twice to go back and select Resume. Your game should start.

12 Tweak your games entries

To hide the games that come with Recalbox, from EmulationStation press Start > Main menu > Games settings > Hide preinstalled games. Unfortunately, you can't pick and choose which get hidden, but you can manually download and re-add any that you'd like to keep.

You can also disable the ports category by editing **recalbox.conf** to include:

```
emulationstation.collection.ports=1
emulationstation.collection.ports.hide=1
```

If you want to add images or change the titles of the games you've added to Recalbox, the easiest approach is to use the built-in scraper. Highlight the game in the menu, press Start > Edit game > Scrape. You can also add your own ratings and keywords in this menu.

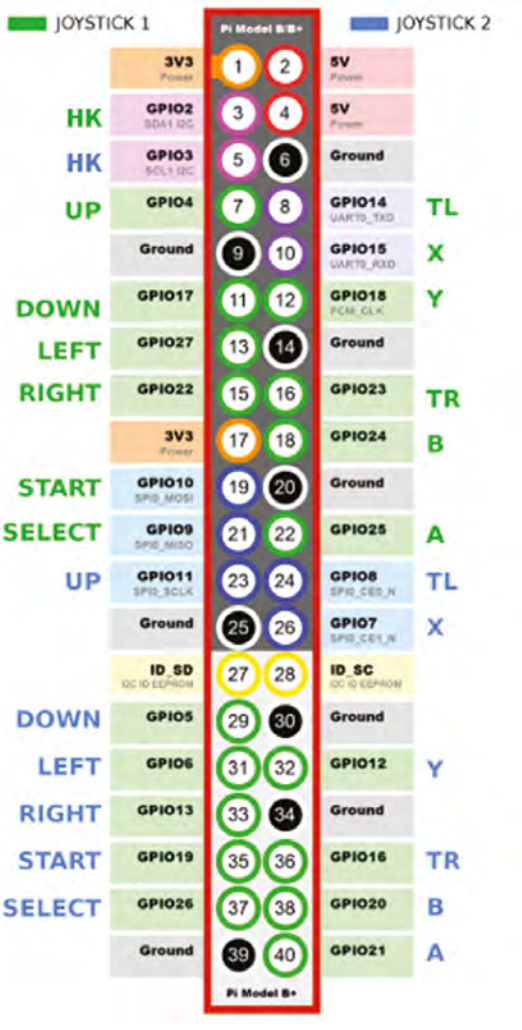
13 Get more games

The creators of the MAME emulator have been given permission to distribute some early arcade games, which you can find for download at **magpi.cc/mameroms**.

Many other emulated arcade games have been released for use on modern computers, but some – including collections by SNK, Capcom, Irem, and Namco – require an additional extraction and re-bundling stage. You can find tools and game lists to help you buy and use these at RED-project (**magpi.cc/redproject**) and SF30ac-extractor (**magpi.cc/sf30ac**). Linux GOG users may also require innoextract (**magpi.cc/innoextract**). Non-Neo Geo arcade games should go into the **roms/MAME** directory.

The homebrew scenes for arcade games tend to focus on physical releases, but we've had luck with Codename: Blut Engel for Neo Geo and Santaball (**magpi.cc/neogeohomebrew**) for Neo Geo CD.

For more retro and homebrew games that work well with arcade controls, including Sega's Mega Drive Classics collection, see **magpi.cc/legalgameemu** and **magpi.cc/legalroms**.

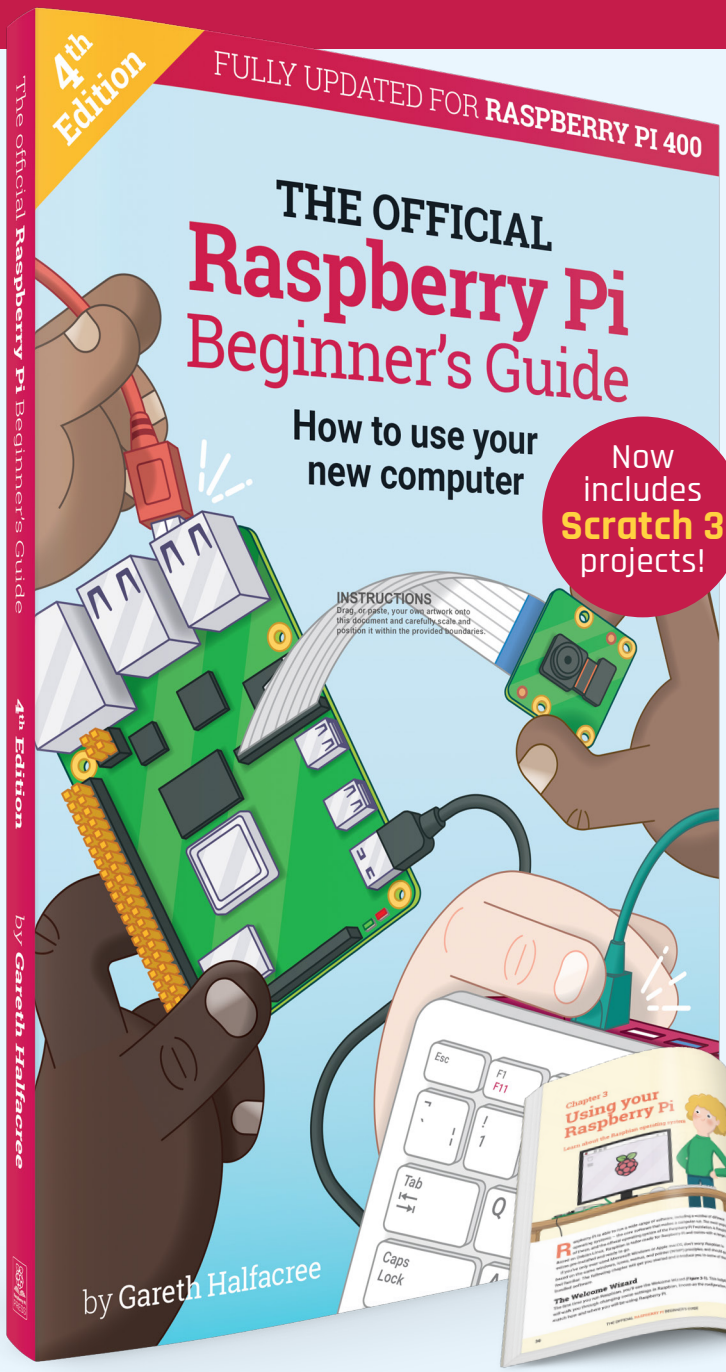


Warning!
Copyright alert!

It is illegal to download copyrighted game or BIOS ROMs in the UK without the permission of the copyright holder. Only use official purchased or freeware ROMs that are offered for download with the consent of the rights holder.

magpi.cc/legalroms

THE OFFICIAL Raspberry Pi Beginner's Guide



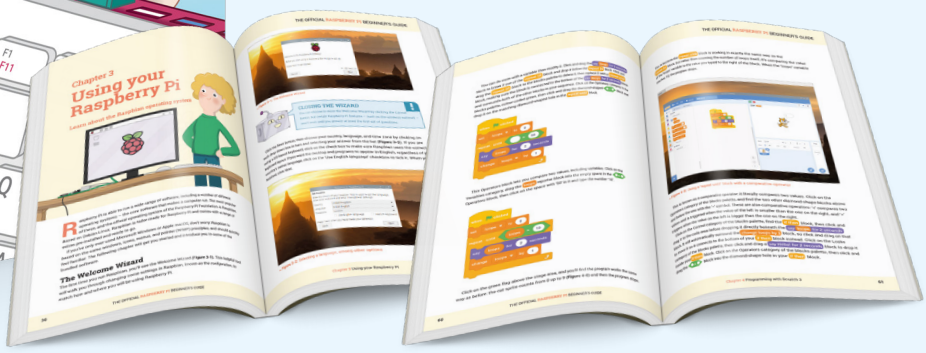
The only guide you need to get started with Raspberry Pi

Inside:

- Learn how to set up your Raspberry Pi, install an operating system, and start using it
- Follow step-by-step guides to code your own animations and games, using both the Scratch 3 and Python languages
- Create amazing projects by connecting electronic components to Raspberry Pi's GPIO pins

Plus much, much more!

£10 with FREE worldwide delivery



Buy online: magpi.cc/BGbook

Part 04

Build an arcade machine: Decorate your cabinet

You've built an arcade cabinet, but vinyl decals and edge moulding will bring it to life

Most arcade cabinet kit suppliers print pre-designed or custom vinyl decals to decorate your cabinet. Third-party printers can produce vinyls to your specification, but make sure that you provide accurate measurements.

Our vinyl decals, bought from Omniretro (**magpi.cc/omniretro**), arrived on a roll and had to be cut out, but some firms will die-cut vinyls for you. We'll use a wet application process, which makes it easier to remove and reposition decals for a short while after initial placement, to help you get a perfect alignment.

Mark up in chalk pen and use a metal ruler to help cut your screen decal to size

01 Flatten your vinyl decals

If your vinyls all came on a single roll, the first step is to cut each of them out. First separate them, if they're on a single roll, but leave generous margins. Spread them out on a table or on the floor and weigh them down – coffee table books and textbooks are good for this. Leave them for at least an hour or two: 24 hours is better.

02 Cutting out

Now they're flat, it's time to cut out your vinyls. Try to get rid of all white matter on straight edges. The easiest way is to line up a long metal ruler so that it just covers the edge of the printing, and run a scalpel down the outside of it. Curved sections for the cabinet side panels are trickier, but you don't need to worry about these as they're easy to trim down once fitted. For now, trim them freehand and leave as much white overmatter as you feel comfortable with.

03 Partial disassembly

Depending on the design of your cabinet, you may need to remove a side panel to take out the acrylic marquee and screen panels. Before doing this, use a liquid chalk pen and ruler to mark the edges of your LCD display on the acrylic, so we can accurately hide the bezel.

If you've previously fitted joysticks and buttons to your control panel, this is the time to remove them too. Apply steady pressure to the rear of snap-in style buttons to pop them out of the cabinet. People with large fingers may find a ButterCade Snap Out Tool useful for this.

04 Applying vinyl to your marquee acrylic

Two acrylic parts require individual application of vinyls: the marquee and the screen that goes in front of your monitor. The former is easy: remove the backing from the vinyl marquee decal and any protective film from the acrylic. Spray both the acrylic and the adhesive back of the vinyl with two or three squirts of application fluid. You want them to be damp all over but not awash.

Pick up the vinyl decal in both hands and, starting at one end of the acrylic, line it up with the edges and paste it down. If you're not happy with the positioning, firmly hold the vinyl and snap it back up – the application fluid will help it release easily.

Once it's positioned, use your applicator and a cloth to smooth it down, drive out any excess water, and remove any trapped air bubbles under the vinyl. Trim any excess vinyl spilling off the edge of the acrylic with a knife.

05 Measuring your screen acrylic

Cutting your screen decal to size is awkward. Before removing the screen acrylic from the cab, we marked the inner position of our monitor's bezel on the acrylic using a chalk pen. If your cabinet has a detachable VESA mount, bring the monitor with you to help line everything up.

> "Grab your screen vinyl and mark up the area to cut out"

Measure the distance between the edge of the acrylic and the chalk line you drew on it. Measure in multiple places to be sure of distances. Our 24-inch monitor's positioning and bezel size means that we cut 35 mm in at the top and sides, and 65 mm from the bottom – yours will differ.

06 Cutting your screen decal

Once you've taken the measurements, grab your screen vinyl and mark up the area to cut out. Mark on the side showing the picture, paying particular care to the corner positions. Double-check these by placing the acrylic on top to make sure both sets of marks line up.

▲ Use a vinyl applicator and a cloth to stick down, remove excess moisture, and eliminate air bubbles from your decals

You'll Need

- Vinyl decals
- U-moulding/ T-moulding
- Scalpels/craft knives
- Strong scissors
- Liquid chalk marker pen
- Metal rulers, tape measures
- Vinyl application fluid
- Vinyl applicator
- Neoprene glue

Grab your metal ruler, place it along your marked line, and cut a rectangle out of the middle of the vinyl decal with a blade. If in doubt, err towards leaving too much vinyl rather than too little. To check positioning, put the acrylic over your monitor, and your vinyl over the acrylic: they should all line up.

07 Screen decal application

Now, turn the vinyl upside down, remove its backing, spray it and the acrylic with the application solution, and stick it down using an applicator and cloth. Residual chalk marks can be wiped off using a bit more of the application solution.

Allow both the marquee and screen decals to dry for a day, trim them if needed, slide them back into your cabinet, and reattach anything you removed. This will probably be the last time you do this, so make sure the side panels are on securely and are correctly lined up and bolted to your stand, if you have one.

If you plan on back-lighting your marquee, this is a good time to put in your light. We used adhesive tape and supplied clips to mount a 50 cm USB-powered LED light on the underside of the marquee, just in front of the speakers.

08 Applying flat vinyls

If you have a full-height cabinet or a bartop and stand, you'll probably have a number of flat, front-facing areas to decorate – in our case, the front cupboard door of our stand, its base, and the front of its foot. Do these next to get your hand in.

The drill is the same for all of them: place the vinyl decal face-down on the floor, remove its backing, spray both it and the surface you're

▶ We marked the inner position of our monitor's bezel on the acrylic using a chalk pen

applying it to, position your decal, and smooth it out with your applicator. Use a scalpel to trim off any overmatter. For the door, we applied the decal with the door in place – knob removed, starting at the top. We had to open the door to flatten and trim the vinyl in places.

09 Control panel decals

Most control panel decals wrap around the top and front of your panel. Buttons and joysticks should not be present during application. This is a relatively easy section to apply, but watch your position if there are decorative patterns designed to surround specific buttons or joysticks.

You may need to trim overmatter from the sides with a scalpel to get the decal to fold over the front face properly. Be careful when smoothing the vinyl on this fold, as it can be prone to both trapped air bubbles and damage from the join beneath.

10 Cabinet positioning

Side panels are the largest pieces of vinyl you'll be applying, but they're less intimidating than they seem. For a standalone bartop, one person can mount them in a vertical position with little fuss, as shown in Omniretro's video at **magpi.cc/omniretrovinyl**.

Full-height cabinets present more of a challenge due to their height and the size of the vinyl – a second person is useful here. You can apply long vinyls in an upright position, but we'd already attached rubber feet to our cabinet, so we used these to help pivot the cab down to lie on a sheet of cardboard on the floor.

11 Apply side panel vinyls

Lying flat and sprayed down as before, it's easy to line up the side-panel decal. Make sure everything's covered – with two people, it's easy to snap the decal back up if you make a mistake, then use a cloth and applicator to drive out excess moisture. Use a Stanley knife to trim the vinyl to size – its solid metal body makes it easy to follow the line of the cabinet's curves.

Go around again to remove any air bubbles and ideally leave the vinyl to dry for at least a couple of hours before pivoting the cabinet back up and lowering it to expose the opposite side. Repeat the process.

▲ You can leave some white-space overmatter on side panel decals before application, as they're easy to trim with a knife afterwards

> "Side panels are the largest pieces of vinyl you'll be applying"

If your cabinet has separate stand and bartop parts, but uses a single sticker, there will be a slight ridge where these join. However, careful application (and a sympathetic vinyl design) makes this effectively invisible. Just be careful smoothing around it.

Top Tip

Screen materials

Acrylic scratches really easily, so tinted tempered glass is an excellent alternative for your cabinet screen.

▲ Highly flexible, U- and T-moulding are used to give a clean finish to the cabinet edges

▲ Demonstrated here without glue, flex U-moulding backwards and use a finger or thumb to press it into place on a cabinet edge

Warning!
Solvents

Always use solvents in a well-ventilated area and keep away from open flames.

magpi.cc/solvents

12 Moulding

We used U-moulding on our cabinet, with neoprene glue to hold it in place securely. First, measure and use scissors to cut two strips to go above and below the marquee – it's better to cut these a few millimetres too long and then trim than it is to have a gap.

Use a spatula to help apply neoprene glue along the edge you're working on, then use the tube's nozzle to apply glue to the inside of the U-moulding.

To lock U-moulding into place, bend it backwards to spread the U-shaped section, push that onto the edge you're applying it to, and then roll the moulding down along the edge, using a finger to push it into place.

When applying it to a long section, such as each side of your cabinet, start at the front underside – rubber feet help access here – apply glue to the cabinet edge and the first 50 cm of your roll of moulding, and have someone else feed it to you as you work up and around the cabinet. When you get to the bottom at the back, cut off your moulding with scissors.

T-moulding locks into a pre-cut groove along the edges of your cabinet, making it more secure, but it's still a good idea to apply glue to the flat surfaces for security. Either way, use a rubber mallet to gently tap down your moulding at the end.

You can use acetone to clean the glue off your hands and the moulding, but keep it away from the vinyl.

After spraying the vinyl decal, and the acrylic, with our homemade fluid, we applied it and smoothed down with an applicator and cloth

Vinyl application fluid

You can buy commercial vinyl application fluid (**magpi.cc/vinylfluid**), widely used by car customisation enthusiasts to apply decals, but we filled a spray bottle with the following homemade formula:

- 66 ml surgical spirit
- 132 ml water
- 2 drops washing up liquid

You can use warm water with a drop of washing up liquid alone, but the surgical spirit reduces drying times, which means less waiting between different stages of application and decorating.

13 Finishing moves

Use a scalpel to cut out the vinyl above the button holes: locate a hole, pierce it with the blade, slice until you find the edge of the hole, and then follow the hole round to remove all the vinyl. Do this for all your joystick and button holes.

As described in *The MagPi* #105 (**magpi.cc/105**), screw your joysticks back into place from the inside. If you're going to put protective acrylic panels over your control panel, this is the time to do it – they're held on solely by the buttons.

However, because our cabinet is for home use, we've left the vinyl bare for a more comfortable and attractive finish. If your cabinet will see lots of play, acrylic will protect it and cut down on wear and tear. Whichever you choose, connect a DuPont cable to each button and pop them into place.

Follow the instructions from issue 106 to connect your buttons and peripherals to Raspberry Pi.

Warning!
Sharp objects

Take care when using knives and scalpels.

magpi.cc/handknives

Part 05

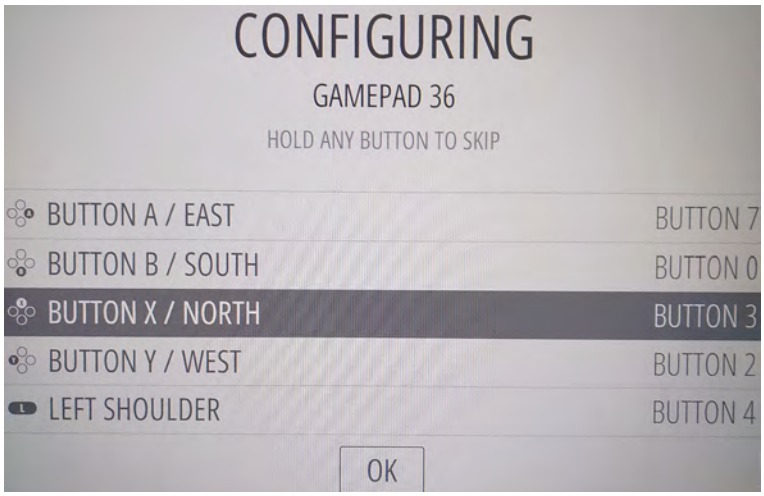
Build an arcade machine: RetroPie and stream from Steam

Use RetroPie as your arcade operating system and add extra emulators with support for Steam Link. Stream games from a powerful PC to Rasbperry Pi

Last month, we used Recalbox for our main arcade cabinet operating system, but it's not your only choice. In this final instalment of the 'Build an arcade machine' series, we'll use the RetroPie distribution, currently at version 4.7, to provide extra features such as Steam Link support, as well as taking a longer look at where to buy arcade games and how to get them onto your system. This tutorial assumes that you already have a fully assembled and wired arcade cabinet.

01 Install and prepare RetroPie

Fire up Raspberry Pi Imager, connect your microSD card writer, and install RetroPie from its Choose OS menu. Re-mount the microSD card once you've finished flashing it, because we've got some changes to make.



▼ More button assignments are available on RetroPie than you have arcade controls. You can skip the ones that don't match up

As with our DB9 joystick project in issue 101 (**magpi.cc/101**), we have to tell the GPIO to treat the controls as pull-up switches. Recalbox, by comparison, implements this by default.

Create the **pullup.sh** file we've supplied (**magpi.cc/pullupfix**). You can put it anywhere you like – we stuck ours in **/home/pi/**. Now open **/etc/rc.local** on the SD card and, above the `exit` line, add:

```
/home/pi/pullup.sh
```

This will load your pull-up settings on boot. If you're setting up your disk on a Linux system, you can set **pullup.sh** as executable now. Otherwise, we'll do that on first boot.

02 First boot

Make sure you have a keyboard plugged into your cabinet for this bit. We left ours propped up against the marquee acrylic during setup for easy access. A Bluetooth keyboard is a viable alternative, but it's easier to start with a wired connection.

Plug in Raspberry Pi's power. It should boot to the EmulationStation interface, but we can't configure the controls until we've set our pull-up script executable. Press **F4** to exit to the command line and type:

```
chmod /home/pi/pullup.sh +x
```

While we're here, let's enable SSH:

```
sudo raspi-config
1 system option
s3 password
```

Raspberry Pi is powerful enough to run most games, and you can stream classic and modern arcade titles from a separate PC on your network via Steam Link

Steam Link lets you create a dedicated control map. Remember to map your hotkey button to its guide button

DOWNLOAD THE FULL CODE:
magpi.cc/rpipullupfix

```
enter a new password
3 interface options
enable ssh
yes
```

You can now SSH into Raspberry Pi from another PC using a client such as Remmina or PuTTY.

03 Add hotkey button support

If, like ours, your arcade cabinet's GPIO controller setup has either one or two extra hotkey buttons for easy access to save, load, and exit shortcuts while playing, then the standard version of the mk_arcade_joystick_rpi driver available from RetroPie's package manager won't support them. We'll have to manually add an updated version from maintainer Recalbox's GitLab repo.

At the command line, type:

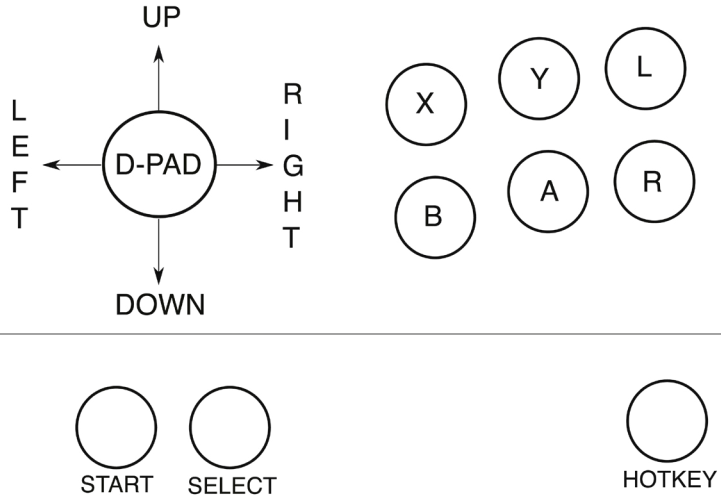
```
git clone --branch v0.1.9 https://gitlab.com/recalbox/mk_arcade_joystick_rpi.git
sudo mkdir /usr/src/mk_arcade_joystick_rpi-0.1.9/
cd mk_arcade_joystick_rpi/
sudo cp -a * /usr/src/mk_arcade_joystick_rpi-0.1.9/
nano /usr/src/mk_arcade_joystick_rpi-0.1.9/dkms.conf
```

In this file, change `PACKAGE_VERSION="$MKVERSION"` to `PACKAGE_VERSION="0.1.9"`. Press **CTRL+X** to exit, then **Y** to save.

Back at the command line, enter:

```
sudo dkms build -m mk_arcade_joystick_rpi -v 0.1.9
sudo dkms install -m mk_arcade_joystick_rpi -v 0.1.9

reboot
```

> We'll use the RetroPie distribution to provide extra features such as Steam Link support



▲ Button and joystick correspondences for player controls; we recommend this configuration for use with RetroPie. L and R map to the right and left shoulder buttons

04 Optional: Load your hotkey driver

When Raspberry Pi has rebooted, SSH back in and type:

```
sudo modprobe mk_arcade_joystick_rpi map=1,2
```

Go over to the arcade machine and press **F4** to get to the command line and test your controllers:

```
jstest /dev/input/js0
jstest /dev/input/js1
```

If that works, it's time to load that module on boot. At the command line:

```
sudo nano /etc/modules
```

In this file, add the following on a new line, then save and exit.

```
mk_arcade_joystick_rpi
```

Next, at the command line:

```
sudo nano /etc/modprobe.d/mk_arcade_
joystick.conf
```

In this file, add the following:

```
options mk_arcade_joystick_rpi map=1,2
```

Now save, exit and reboot.

05 Configure RetroPie

There's a bit more configuration to do before RetroPie is ready to go. SSH in and type:

```
sudo ~/RetroPie-Setup/retropie_setup.sh
```

…to open the ncurses configuration menu.

If you did not manually install a hotkey version of the mk_arcade_joystick driver in the previous steps and do not need one, go to:

```
P manage packages
driver
819 mkarcadejoystick
```

…and install it.

If you need to connect any Bluetooth keyboards or controllers, go to:

```
C Configuration / tools
804 bluetooth
```

Press **R** to register a device and follow the pairing instructions.

832 samba in the configuration menu sets up Samba shares so you can easily transfer ROMs and BIOS images over your local network

You can add extra emulators here, but we'll come to that later. For now, select the R Perform reboot option from the main menu.

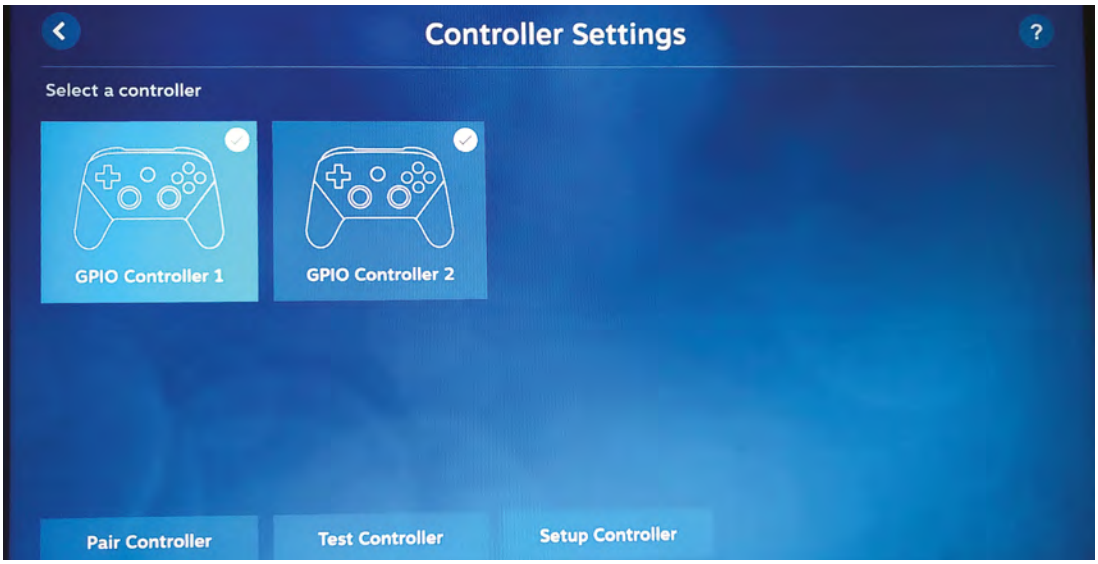
06 Input configuration

When RetroPie reboots, it should inform you that it can detect two GPIO controllers. Press and hold any button on the left-hand button bank to configure controls for player 1. Because arcade controls don't map perfectly to a gamepad, you'll have to skip some buttons by pressing and holding any key.

> When RetroPie reboots, it should inform you that it can detect two GPIO controllers

Map up, down, left, and right on the arcade stick to the D-pad. Follow our button assignment diagram to map the top row to button Y, X, and L(eft shoulder), and the button below to buttons B, A, R(ight shoulder).

Map Start to player 1's left-hand front function button and Select to their right-hand front function button – this will be their 'insert coin'



You'll probably want to reconfigure your controls in Steam Link to better match its Steam Controller-based expectations

button. In our wiring configuration, our single hotkey button – the last we set – is associated with player 1.

Approve your configuration, then set up player 2's controls in the same way.

07 Getting to know RetroPie

With your controllers configured, RetroPie's main interface will open. Press A to select menus and items and B to go back. Press Start to open the main menu and Select to open the options menu. Press the same button again to close each of these.

As you have yet to put any games on the system, only the RetroPie menu will be available. Here, you'll find easy access links to configuration tools, including some we used earlier. Install new emulators and drivers from the RetroPie Setup menu.

You'll probably need to disable overscan to get rid of a black border around the screen. In the ReotroPie menu, select Raspi-config > Display options > Underscan > No and then reboot to solve the problem. Note that button B is mapped to the **ENTER** key in this set of menus.

When you add any new games, ROMs or emulators, you'll have to restart EmulationStation by pressing Select, going to Quit, and then Restart EmulationStation.

08 Install more emulators

Although this is an arcade machine, you can play what you like on it. The core lr-mame2003 and lr-fbneo emulators are included, along with those for popular consoles such as the Sega Mega Drive, used in some arcade systems and for which original games are legally available.

Some emulators require system BIOS images. Sadly, very few of these have been made legally available to emulation enthusiasts. SNK distributes a UniBIOS compatible BIOS set in its 40th Anniversary Collection. We recommend adding the following:

opt > 327 opentyrian – arcade-like DOS shoot-'em-up Tyrian 2.1 is now freeware.

exp > 241 lr-mame – a more up-to-date version of MAME that supports a wider range of ROMs. Install from source for bleeding edge.

exp > 307 digger – a sanctioned remaster of Windmill Software's Dig-Dug.

exp > 334 steamlink – this allows you to stream less emulation-friendly titles directly from a Steam installation on a Windows or Linux PC.

09 Configure your emulator

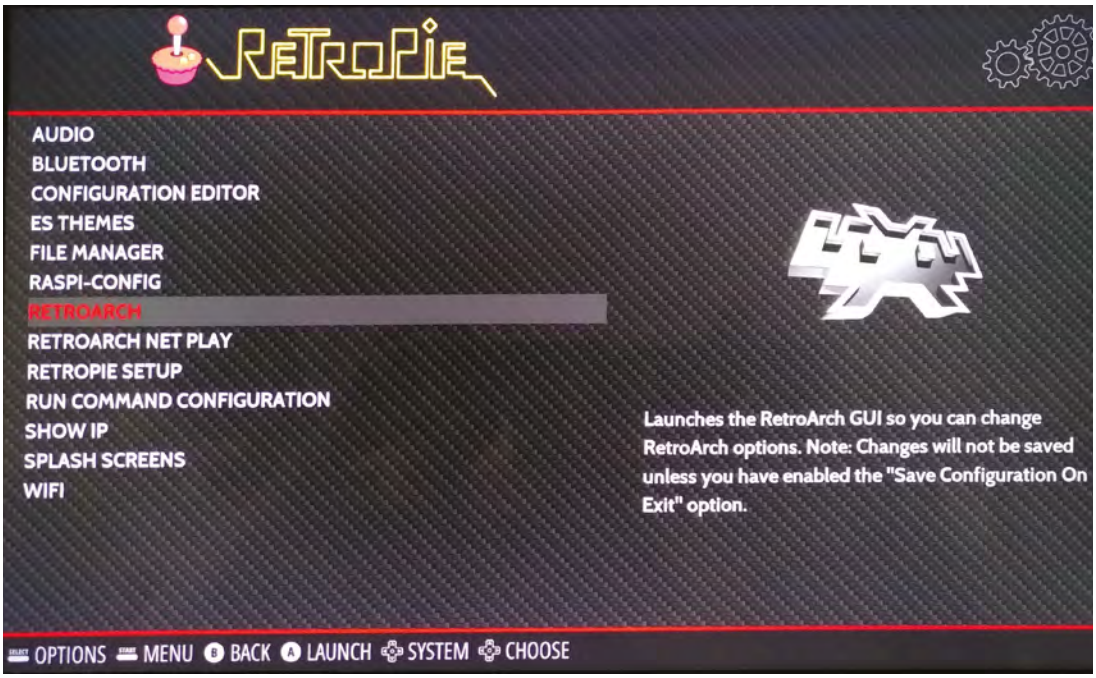
Once you've installed a new emulator, such as lr-mame, you'll have to configure the libretro back end to use it by default for either all games or selected titles. The easiest way to do this is to browse to the game you want to play in the EmulationStation front end.

Go to the Arcade menu, press B to start any game – it doesn't matter if it currently works or not – and then press B again when you're briefly prompted to 'press a button to configure'. Select option 1 to set the default emulator for arcade

Top Tip

Steam Link smoothly

Use a wired Ethernet connection for optimal Steam Link game streaming.



RetroPie is significantly more configurable than Recalbox, although its interface doesn't look quite as slick

Top Tip

A screw loose

If the ball on your joystick is loose, use a screwdriver in the slot on the underside of the stick or cloth-wrapped pliers to hold the shaft still while you tighten the ball.

games and choose lr-mame. Option 2 allows you to select a different emulator for anything that doesn't work well with this.

10 Connect Steam Link

Linking Steam to your arcade cabinet lets you stream a wealth of modern and classic arcade games to Raspberry Pi from a more powerful PC, like Melty Blood, Guilty Gear, Horizon Chase Turbo, and Street Fighter V. After you've installed it and restarted EmulationStation, go to the Ports menu and select Steam Link.

It'll download updates – you will need a keyboard plugged in to approve these – and then run. Make sure Steam is running on a PC on your local network and that Enable Remote Play is ticked under Settings > Remote Play.

On the arcade machine, select the computer you want to link to. Steam Link will show a code. Enter this in your PC's Steam client when prompted. To avoid a resolution mismatch, run Steam with a monitor that matches the resolution of your arcade machine set as your primary display.

11 Configure Steam Link

You may want to reconfigure your controls, as Steam Link doesn't inherit the control layout from RetroPie's EmulationStation, and some games do better with alternative button assignments – for example, to more closely match an Xbox or Steam Controller, which swaps the position of the B and A buttons.

To set these, launch Steam Link, press up to highlight the gear icon, press A (per our button assignment diagram), go right to highlight Controller and press A. Select the controller you wish to configure, then press down and right twice, and select Setup Controller.

Hit the button you want to associate with each Steam Controller-style button as it's displayed on screen. Use a keyboard or your second set of controls to use the skip button at the bottom to bypass extraneous buttons.

12 Why use Steam Link?

Steam Link is an invaluable tool for arcade emulation enthusiasts, not only because you can play more CPU-intensive games, but also because it's the best way of ensuring copyright compliance for a number of re-released arcade games.

We've been playing Ghouls 'N Ghosts from the Capcom Arcade Stadium on our cabinet via Steam Link. Unlike some SNK and Sega re-releases, Capcom doesn't supply emulator-ready ROM files and the EULA for that compilation doesn't allow you to extract its PAK files.

Neil Brown of decoded.legal opines (**magpi.cc/romextractionlegal**) that "when even a legitimate Steam purchaser extracts the ROMs and runs them on their own Raspberry Pi, they infringe Capcom's copyright", making streaming these titles your best option for fully legal home arcade action.

Wireframe

Join us as we lift the lid on video games

Visit **wfmag.cc** to learn more

Travel in time

K.G. Orphanides wants you to play and program imaginary retro games

People get excited about old games. It's not exactly a secret, given the ongoing boom in remakes, remasters, and repackaged classics. But there's something special about trying to come as close as you can to the original gaming experience.

Perhaps the most surprising thing I've found over years of building emulation systems is how broad their appeal is. Kids who've never known a time without HD screens and realistic 3D take to pixelated platformers and EGA edutainment software without a blink.

> Pixel art is still going through a years-long renaissance

Even the difficulty spikes of old-school arcade classics, designed to separate players from their money as efficiently as possible, become artful punctuation in the language of play, thanks to the save states and infinite credits of an emulator-based arcade cabinet.

With greater access to development tools, approachable languages, and supportive communities, game creation is also being radically democratised. And just as there are many people who love to play retro games, there are plenty of individuals and development teams who've set out to create their own.

This can go as far as full-scale releases on physical cartridges and major digital platforms, particularly when it comes to the rich world of Sega Mega Drive and other classic consoles, but both the aesthetic and the inherent technical limitations of developing for older hardware have appeal.

Technical limitations in resolution, memory size and audio voices can make your game simpler to program. This has directly led to the rise of Fantasy Consoles, virtual 8-bit machines that run on your computer. Priced at $15 (£11), Pico-8 (magpi.cc/pico8) is the best known of these, with official support for Raspberry Pi and a great community.

Pixel art is still going through a years-long renaissance, with many artists deliberately adopting 256-, 16- and even four-colour palettes. You'll see these choices reflected in games as disparate as Unpacking (through its EGA camera filter) and The Eternal Castle, a 'remake' of a game that never existed. If you want to start making pixel art on Raspberry Pi, Aseprite (**magpi.cc/aseprite**) can be compiled, and an unofficial ARM build of Pixelorama (**magpi.cc/pixelorama**) is now available.

More modern retro aesthetics can be found in the low-rez horror games exemplified by the Haunted PS1 Demo Disc series (**magpi.cc/hauntedps1itch.io**).

Although Unity and Unreal Engine don't run on Raspberry Pi, the impressive Godot engine does (**magpi.cc/godot**), albeit with a few limitations.

We've got you covered if you want to begin your journey into retro game programming right here with PyGame Zero (**magpi.cc/pygamezero**).

Meanwhile, for those who want to get a feel for techniques from the past and present, BBC BASIC (**magpi.cc/bbcbasic**) is available for Raspberry Pi, while developing with using C++ and SDL is far more fun that it has any right to be.

AUTHOR

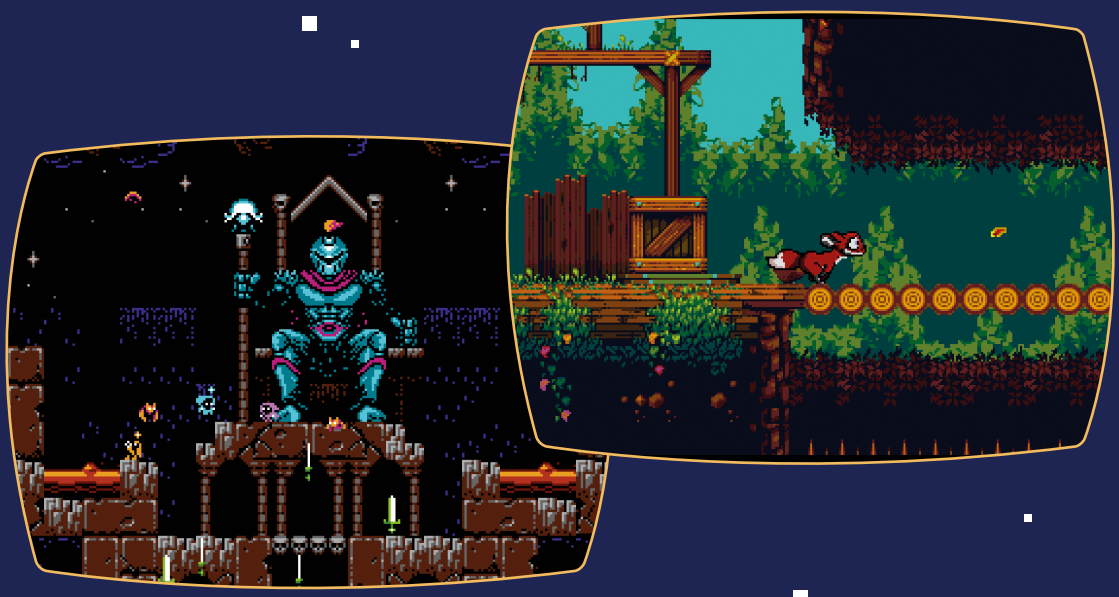
K.G. Orphanides

K.G. variously makes weird games, DIY emulation consoles, music, documentation, lost software archives and quixotic builds.

@KGOrphanides

game on

VILROS



Retro Gaming with Raspberry Pi shows you how to set up a Raspberry Pi to play classic games. Build your own games console or full-size arcade cabinet, install emulation software and download classic arcade games with our step-by-step guides. Want to make games? Learn how to code your own with Python and Pygame Zero.

- ***Set up Raspberry Pi for retro gaming***
- *Emulate classic computers and consoles*
- ***Learn to code your own retro-style games***
- *Build a console, handheld, and full-size arcade machine*

Price: £10
9 781912 047772
Raspberry Pi PRESS