Minecraft - Pi edition

ISSUE 27 - OCT 2014

Get printed copies at themagpi.com

The MagPi™

A Magazine for Raspberry Pi Users

Game Console Controllers

FUZE BASIC

Matboard

Using Git

Ultrasonic Range Finder

C++ Operator Overloading

Build QR codes in Minecraft

ISSN 2051-9990

The MagPi

http://www.themagpi.com

Welcome to Issue 27 of The MagPi magazine. This month's issue is packed cover to cover with something for just about everyone!

Are you tired of controlling your Raspberry Pi with the same old mouse and keyboard? Have you ever wished you could have the ergonomic feel of a console controller in your hands when playing some of those retro games we have written about in past issues? If you answered yes to either of these questions, why not take a look at Mark Routledge's fantastic article describing how to do just that.

Alec Clews talks us through the use of Git, a free version control software package that we also use here at The MagPi to ensure that all of the team work on the most up to date copy of each issue. This is a great read, especially if you work with any type of document or file as part of a team.

As you can see from our front cover, we return to the popular world of Minecraft in Dougie Lawson's clever article on building QR code structures inside the game. We also have more physical computing from ModMyPi, and a great father and son story on building and funding a Raspberry Pi project through Kickstarter.

Of course we have not forgotten about programming. William Bell continues his popular C++ series and we also have part three of our game programming series using FUZE BASIC. Start thinking of some game ideas now because in the next issue we will have a game programming competition.

If you want even more from The MagPi this month then why not join us on the 11th October at the SWAMP Fest event (see this month's Events page) where we will have our own stand. We look forward to seeing you there.

We hope you enjoy this month's issue and don't forget to like our Facebook page and leave a comment at http://www.facebook.com/MagPiMagazine.

Chief Editor of The MagPi

The MagPi Team

Ash Stone - Chief Editor / Administration
Ian McAlpine - Issue Editor / Layout / Proof Reading
W.H. Bell - Administration / Layout
Bryan Butler - Page Design / Graphics
Matt Judge - Website / Layout
Aaron Shaw - Layout
Nick Hitch - Administration
Colin Deady - Layout / Proof Reading
Dougie Lawson - Testing
Nick Liversidge - Proof Reading
Age-Jan (John) Stap - Layout

Contents

4 **CHOOSE YOUR WEAPON**
Connecting an XBOX360, PS3 or Wiimote controller to a Raspberry Pi

12 **THE MATBOARD PROJECT**
A story of Kickstarter, GPIO and water buckets

18 **PHYSICAL COMPUTING**
Part 3: Using an HC-SR04 ultrasonic range finder

23 **THIS MONTH'S EVENTS GUIDE**
Bristol UK, Swansea UK, Huddersfield UK, London UK

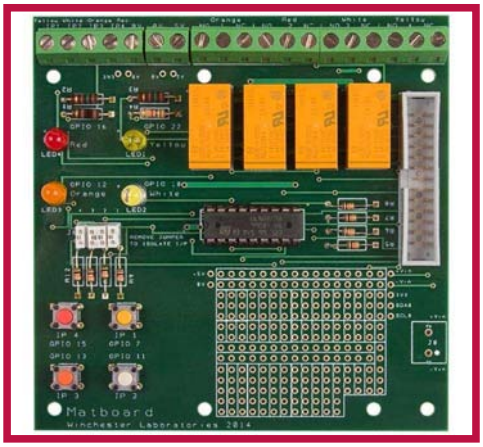
24 **MINECRAFT PI EDITION**
Build QR Code structures inside Minecraft

28 **VERSION CONTROL**
Part 1: Version control basics using Git

34 **FUZE BASIC**
Part 3: Keyboard input, animation and arrays

42 **C++ CACHE**
Part 7: Operator overloading

48 **HAVE YOUR SAY**
Send us your feedback and article ideas



CHOOSE YOUR WEAPON
Adding console game controllers

Connecting an XBOX360, PS3 or Wiimote controller to a Raspberry Pi

Mark Routledge

Guest Writer

SKILL LEVEL : INTERMEDIATE

Choose your Weapon

It is possible, and lots of fun, to use a variety of today's modern console controllers for your Raspberry Pi projects. They are quite easy to install, readily available and you get quite a bit of kit for your buck! This article describes how to setup a Raspberry Pi to use a Microsoft® XBOX 360 gamepad (wired and wireless), a Sony® PS3 gamepad and Nintendo® Wiimote.

For wireless devices you will need either the XBOX gamepad wireless adapter, or for the Wiimote and PS3 gamepad you can use an expensive bluetooth dongle. See http://www.elinux.org/RPi_USB_Bluetooth_adapters for a list of known working bluetooth dongles.

I have tested all the code on a Raspberry Pi Model B, as well as on the new Model B+ board.

Before installing any of the software ensure you are using a recent version of Raspbian and have updated your system. At the command line enter:

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

To test all of the hardware, an easy piece of software to use on the Raspberry Pi is `jstest-gtk` and the standard `joystick` library.

Enter the following command to install the software:

```
sudo apt-get install jstest-gtk joystick
```

This program should now show up in your desktop ready for you to test. You will find the link under Menu → `Other` → `joystick testing and configuration tool`.

Once installed it should display your hardware for example: Connecting your hardware.

A brief note about using the bluetooth adapter. I have found these can be a little tempremental at times, so I strong recommend connecting your dongle directly to the Raspberry Pi and connect any other hardware via a powered USB hub, e.g. the excellent Raspberry Pi Hub! The bluetooth dongle will work from a powered hub on a low level, but the bluetooth stack may not be able to see it.

Setting up the XBOX gamepad

First open up LXTerminal and install the required driver for the XBOX 360 gamepad. The same driver is used for both the wired gamepad and wireless gamepad, using the adapter. Enter:

```
sudo apt-get install xboxdrv
```

It is possible to now use the XBOX gamepad with a variety of Linux games by using:

```
sudo xboxdrv --silent
```

However, this does not really give you much control over what each axis / button does. You can use the joystick testing and configuration tool mentioned earlier but it can be difficult to get setup. Luckily it is possible to configure the XBOX gamead in a variety of ways. Each configuration requires a setup (text) file which you provide when launching the gamepad and driver. Through this you can turn one of the analogue sticks into a mouse and the buttons into keypresses etc. This will make it usable with programs like Minecraft, Doom, Quake3 plus all known emulators. The possibilities are endless, and exciting.

Let us make a directory to store these setup files, which contain the button mappings. Enter:

```
cd ~
mkdir XBOX
cd XBOX
```

Create a configuration text file called basic_config. You can use any editor, but nano is quick and simple. Enter:

```
nano basic_config
```

Now enter the following mappings:

```
#----------------------------------------
# This file is the Basic Definition of
# the controller, allowing a deadzone.
# It will run silently.
#----------------------------------------
# Setup the DPad as buttons and triggers.
#
[xboxdrv]
silent=true
deadzone=6000
dpad-as-button=true
trigger-as-button=true
#
#----------------------------------------
# Map the right analog stick as absolute
# values (x2 and y2) and the left analog
# stick as mouse relative (x1 and y1).
#
[ui-axismap]
x2=ABS_X
y2=ABS_Y
x1=REL_X:10
y1=REL_Y:-10
#
# ----------------------------------------
# Map the four coloured buttons a, b, x
# and y and set each one as a different
# key, or equivalent joystick button.
# In this case Left Shift, joystick
# buttons C and A and the key C.
# Map the triggers and bumpers in the
# same way; lt, rt, lb and rb.
# Map the DPad du, dl, dd, dr as WASD.
# Map the Back, Start and Guide (XBOX)
# buttons to Home, Escape and Enter.
#
[ui-buttonmap]
a=KEY_LEFTSHIFT
b=BTN_C
x=BTN_A
y=KEY_C
#
lb=KEY_LEFT
rb=KEY_RIGHT
lt=KEY_Z
rt=KEY_SPACE
#
dl=KEY_A
dr=KEY_D
du=KEY_W
dd=KEY_S
#
guide=KEY_HOME
back=KEY_ESC
start=KEY_ENTER
#--------------- EOF --------------------
```

Save and close the configuration file and exit from nano by pressing the <Ctrl>+<X> keys together, followed by the <Y> key then <Enter>.

Description of the XBOX config file

This file is the basic definition of the controller, allowing a deadzone. It will run silently, meaning it will not output additional information to the terminal.

The file sets up the DPad as buttons and triggers. The next few sections are used to map the right analogue stick as absolute values on the x2 and y2 axes and the left analogue stick as mouse relative values on the x1 and y1 axes.

It maps the four coloured buttons a, b, x and y and sets each one as a different key, or equivalent joystick button, in this case Left Shift, joystick buttons C and A, and the C key.

It also maps the triggers and bumpers (lt, rt, lb, rb) in the same way.

Finally it maps the DPad (du, dl, dd, dr) as the classic W, A, S, D keyboard setup. It also sets the Back, Start and Guide (XBOX) buttons to the Home, Escape and Enter keys respectively.

You can now physically plug in your XBOX gamepad or adapter if you have not already done so. I recommend connecting the gamepad or adapter for the XBOX 360 to a powered hub.

Do a quick list of USB devices to check that the gamepad or adapter has been detected. Enter:

```
lsusb
```

You should see an entry for the XBOX gamepad or adapter similar to,

```
Bus 001 Device 009: ID 045e:028e Microsoft
Corp. Xbox 360 Controller
```

or,

```
Bus 001 Device 008: ID 045e:0719 Microsoft
Corp. Xbox 360 Wireless Adapter
```

Finally call the basic configuration mapping using the `--config` switch with your configuration file:

```
sudo xboxdrv --config ~/XBOX/basic_config
```

You can run this command in the background by adding an & at the end of the command.

At this point you can either use the command,

```
sudo jstest /dev/input/js0
```

to test the gamepad from the command line or,

```
startx
```

and select the joystick testing and configuration tool.

For the wired gamepad you should be connected and up and running.

If you are using the wireless gamepad you will have to sync it to the adapter. This is done by pushing the Sync button on the adapter then pushing the Sync button on the gamepad.

Note: Even though the gamepad is connected, the XBOX pad quadrant light will continue to flash! There is no quick or easy way to turn off the pad once you have finished using it, other than temporarily removing the battery pack for a few seconds.

If you are keen to test your new setup in Python follow the excellent tutorial provided by Rhishi Despanda in The MagPi Issue 26, pages 12-13.

Setting up the PS3 gamepad

If you have the PS3 gamepad wired into a USB hub with the charging cable, the Raspberry Pi should detect the gamepad and work almost immediately. You may have to press the "PS" button to start though.

Alternatively you can use a bluetooth dongle. Again, connect the bluetooth dongle directly into one of the Raspberry Pi's USB ports, not via the USB hub. The following drivers and settings will only work with genuine PS3 gamepads. Cheap imports simply do not seem to work. Save your money and buy an official second hand PS3 gamepad!

Wired PS3 gamepad

To test a wired PS3 gamepad list the USB devices with the command:

```
lsusb
```

You should see something similar to:

```
Bus 001 Device 008: ID 054c:0268 Sony
Corp. Batoh Device / PlayStation 3
Controller
```

Either test from the console with the command,

```
sudo jstest /dev/input/js0
```

or,

```
startx
```

and use the joystick testing and configuration tool.

Wireless PS3 gamepad

Install the required libraries to start bluetooth. This may seem like a lot, but it will save time and bother later. Enter:

```
sudo apt-get install bluez-utils
sudo apt-get install bluez-compat
sudo apt-get install bluez-hcidump
sudo apt-get install checkinstall
sudo apt-get install libusb-dev
sudo apt-get install libbluetooth-dev
```

Next download the drivers for pairing the bluetooth dongle to the PS3 gamepad. Enter:

```
sudo wget http://www.pabr.org/sixlinux/si
xpair.c
gcc -o sixpair sixpair.c -lusb
```

Note: If this second command fails ensure that you have installed `libusb-dev` properly.

After this you should have a executable file called `sixpair`.

Check that your bluetooth dongle is detected by listing the USB devices with the command:

```
lsusb
```

You should see your adapter listed, for example:

```
Bus 001 Device 004: ID 0a12:0001 Cambridge
Silicon Radio, Ltd Bluetooth Dongle (HCI
mode)
```

Now connect your PS3 gamepad via a USB cable to the Raspberry Pi and use sudo to execute the `sixpair` file:

```
sudo ./sixpair
```

If you receive an error about `hcitool` then again make sure you installed all the above packages.

The program should list two MAC addresses, but they may not be the same. For example:

```
Current Bluetooth master:
00:15:83:0c:bf:eb

Setting master bd_addr to
00:1b:dc:0f:ed:b5
```

Run the command again and you should see:

```
Current Bluetooth master:
00:1b:dc:0f:ed:b5

Setting master bd_addr to
00:1b:dc:0f:ed:b5
```

You have successfully paired your controller with your bluetooth dongle. We now need to install the Sixaxis drivers required for the PS3 gamepad to work like a joystick.

Install the Sixaxis joystick manager

We will download the latest archive and compile it. To get the driver tarball (compressed file) enter the following command all on one line:

```
sudo wget http://sourceforge.net/projects
/qtsixa/files/QtSixA%201.5.1/QtSixA-1.5.1
-src.tar.gz
```

Expand the driver with the command:

```
tar xfvz QtSixA-1.5.1-src.tar.gz
```

Naviagate into the driver directory:

```
cd QtSixA-1.5.1/sixad
```

Make the required driver for the Raspbian build:

```
sudo make
```

Make a directory to store different profiles for the PS3 gamepad with the command:

```
sudo mkdir -p ~/var/lib/sixad/profiles
```

Finally, install the required driver that you have just made. Enter:

```
sudo make install
sudo checkinstall
```

The last command automatically creates a package for you, so you can easily uninstall it later if you no longer need it or want to use a different system. To uninstall, enter the command:

```
sudo dpkg -r sixad
```

Now to test it, launch temporary a sixad daemon.

```
sudo sixad --start
```

If the gamepad is still plugged in, unplug it and then press the "PS" button on the Dualshock gamepad. If you feel a vibration then it works. Congratulations!

You may not feel a vibration, but it should display some sort of connection on the screen. For example, you should see somethink like:

```
sixad-bin[6860]: Connected Sony Computer
Entertainment Wireless Controller
(04:76:6E:F1:74:3E)
```

Note: On both the PS3 gamepads that I have tested, NEITHER vibrated! Although both are correctly detected using the `joystick testing and configuration tool`, one of the gamepads shows no signs of connection (no red flashing LEDs) while the other continuously flashes! Both are offical "Dualshock 3" Sixaxis gamepads, but both are also second-hand and I cannot vouch for the working vibration as I do not have a PS3!

To make the `sixad` daemon run automatically every time you start the Raspberry Pi, enter the following command:

```
sudo update-rc.d sixad defaults
```

To end the connection with your gamepad once you are finished using it, run the following two commands:

```
sudo sixad -stop
sudo service bluetooth stop
```

This will free up your PS3 gamepad and it should return to sleep!

Nintendo Wiimote

You will need to use a bluetooth dongle. It is recommend to always plug the bluetooth dongle directly into one of the USB ports on the Raspberry Pi, not the USB hub!

Ensure that the bluetooth driver is correctly installed with the command:

```
sudo apt-get install bluetooth
```

To check the status of the bluetooth driver, enter the command:

```
sudo service bluetooth status
```

You should see something similar to:

```
[ ok ] bluetooth is running.
```

If you do not see this then enter the command:

```
sudo service bluetooth start
```

Now install the CWiiD, WMInput and WMGui packages. These are required to use a Wiimote:

```
sudo apt-get install python-cwiid
sudo apt-get install wminput wmgui
```

Run the `hcitool` to check the bluetooth adapter:

```
sudo hcitool dev
```

Start the Wiimote in detection mode by either

pressing the small red sync button or holding buttons 1 and 2 on the Wiimote until the lights flash. Now run the `hcitool scan` command to locate your bluetooth Wiimote. Enter:

```
sudo hcitool scan
```

After a short period of time it should list your Wiimote as a Nintendo RVL. For example, you should see something like:

```
Scanning ...
00:1D:BC:FB:79:F0       Nintendo RVL-CNT-01
```

Write down whatever was displayed as the MAC address for your Wiimote. We will use this later.

Universal input needs to be setup so it can be used by users other than 'root' users. We will edit the Wiimote rules using nano. Enter:

```
sudo nano /etc/udev/rules.d/wiimote.rules
```

Add the following line to the bottom of the file:

```
KERNEL=="uinput", MODE:="0666"
```

Save and close the file by pressing <Ctrl>+<X> together, then press <Y> and <Enter>.

You need to reboot your Raspberry Pi for the changes to take effect. Enter the command:

```
sudo reboot
```

Wiimote configuration file

We have setup and detected all the hardware. We must now create a custom Wiimote configuration file using nano that maps the inputs from the Wiimote to buttons. Enter:

```
nano /home/pi/mywminput
```

Now enter the following mappings:

```
# -----------------------------------------
# Setup for standard Wiimote
Wiimote.A = BTN_A
# The trigger is button B.
Wiimote.B = BTN_B
# Setup the DPad
Wiimote.Dpad.X = -ABS_Y
Wiimote.Dpad.Y = ABS_X
# Setup the remaining buttons
Wiimote.Minus = BTN_SELECT
Wiimote.Plus = BTN_START
Wiimote.Home = BTN_MODE
Wiimote.1 = BTN_X
Wiimote.2 = BTN_Y
#--------------- EOF --------------------
```

Save and close the file by pressing <Ctrl>+<X> together, then press <Y> and <Enter>.

Finally it is time to test the Wiimote. Enter the following command on one line:

```
sudo wminput -d -c /home/pi/mywminput
<Wiimote address> &
```

Replace the <Wiimote address> with the MAC address we wrote down earlier.

The & at the end just means that this process will keep running in the background while you do other things... like play games!

Finally, test the Wiimote with the command:

```
sudo jstest /dev/input/js0
```

Game on!

Now that you have successfully got your controller working why not take it for a spin using one of the games previously mentioned, setup an excellent emulator like Picodrive or try some Linux games that can be easily installed. Try any of the following games for some light relief:

```
sudo apt-get install abuse blobwars
sudo apt-get install frozen-bubble geki3
sudo apt-get install hex-a-hop ltris
sudo apt-get install supertux triplane
sudo apt-get install xblast-tnt xracer
sudo apt-get install xscavenger
```

I intend to write a follow-up article where I will give examples of how to use each of these different devices from within Python and Linux.

For all this and **much more** please visit my Raspberry Pi documentation project at http://goo.gl/EEQ5J *[Ed: I have added this page to my favourites!]*

Expand your Pi

Stackable Raspberry Pi expansion boards and accessories

ADC-DAC Pi

2x 12 bit analogue to digital channels and 2x 12 bit digital to analogue channels.

Serial Pi

RS232 serial communication board. Control your Raspberry Pi over RS232 or connect to external serial accessories.

ADC Pi

8 channel analogue to digital converter. I²C address selection allows you to add up to 32 analogue channels to your Raspberry Pi.

1 Wire Pi

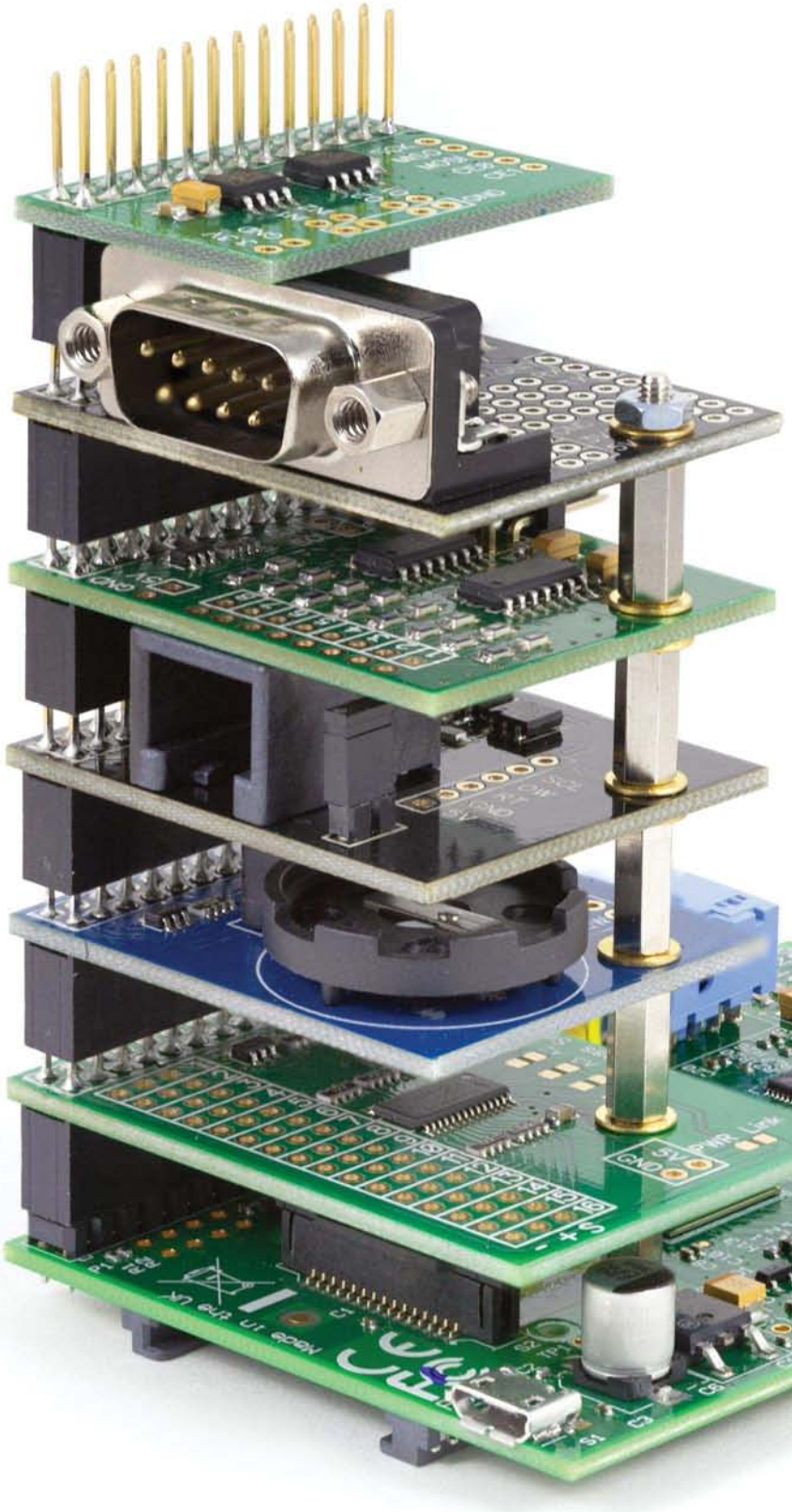
1-Wire® to I²C host interface with ESD protection diode.

RTC Pi

Real-time clock with battery backup and 5V I²C level converter for adding external 5V I²C devices to your Raspberry Pi.

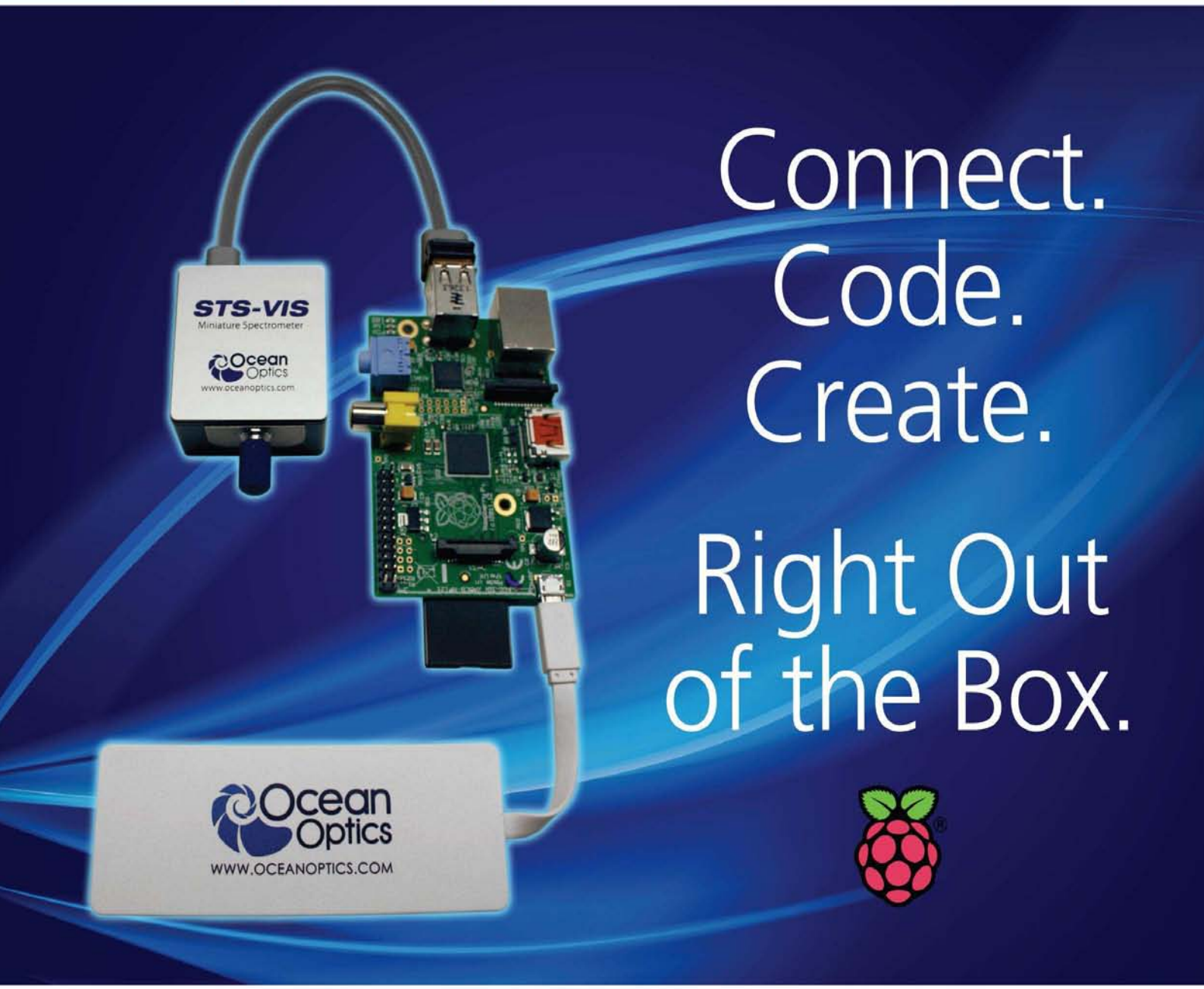
Servo Pi

16-channel, 12-bit PWM controller suitable for driving LEDs and radio control servos.



AB electronics UK

www.abelectronics.co.uk



Connect. Code. Create. Right Out of the Box.

Powered by Raspberry Pi, the STS Developers Kit is the best way to integrate spectral sensing into your application.

Measure color more accurately than the human eye, monitor solar UV levels, prototype compact medical self diagnostic tools or monitor crops from a UAV. The STS Developers Kit makes it easy with integrated device drivers and an easy to access API to get going quickly.

Ocean Optics

www.oceanoptics.com | info@oceanoptics.com | **US** +1 727-733-2447 **EUROPE** +31 26-3190500 **ASIA** +86 21-6295-6600

THE MATBOARD PROJECT
Father and son team develop a new product

A story about Kickstarter, the GPIO and water buckets

Matthew and Martin Brabham
Guest Writers

The beginning

When my son Matthew was 8 years old he was introduced to the Makey Makey in school by a team of engineers from IBM that were doing community work. He loved the way it was easy to develop software and connect to the real world. He even got a trip to IBM where he was shown the Raspberry Pi. As we talked about this at home, I began to tell Matthew about my own experiences with the ZX81 and ZX Spectrum. I think these conversations got both of us interested in getting more involved in engineering. I had completed an engineering degree and had worked for an electronics manufacturer as part of the product development team for many years, but I had not written a single line of code since I left university.

The Raspberry Pi arrived on Matthew's 9th birthday, complete with all the necessary accessories and a blank SD card. To be honest, having not done any research, the first few days were disappointing. It took us most of the day to install Linux and make an LED go on and off on the expansion board we had purchased.

This was no ZX Spectrum - it was a powerful and complex machine.

After playing a few of the Python games, we realised we wanted to do some control projects. Matthew started searching the internet for "GPIO tutorials" and soon we were off to our local electronics store to get some LEDs and switches.

The birth of an idea

From there we started controlling a few motors with relays and somewhere along the way we came up with the idea for a simple, easy to use GPIO expansion board that could be used by just about anyone.

We started on veroboard and then, after explaining to Matthew how PCBs are designed, we decided to go ahead and design one ourselves.

Having looked around for software, we came across the DesignSpark package from RS components. This was great as it is was free, easy to use and came with a library of basic components. See http://www.designspark.com.

The first idea we had was that the board should have four coloured LEDs and switches so that a first time user without doing anything complicated could start to write easy programs; simulating a traffic light, for example. This would

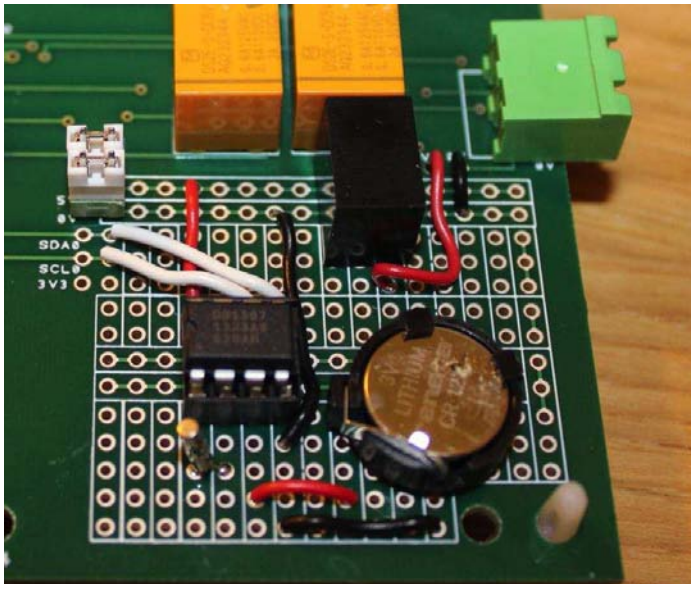
make the board suitable for young users just getting into computing.

However we also wanted people to be able to go a bit further, so we included in the design four relays and switch inputs, so that we could control and read things in the real world.

As the Raspberry Pi has 3.3V logic and can only supply a relatively small amount of current, we used an Darlington pair buffer IC (ULN2803A) to turn on the relays and also to protect the Raspberry Pi from damage.

Sourcing the components presented many challenges, for example finding coloured switches, white LEDs and relays with low power consumption.

We then had the idea to include a prototyping area with the Raspberry Pi I²C (SDA/SCL) pins and also space for a power input connector. We hoped that this would make the board interesting for both beginners and also slightly more advanced users.



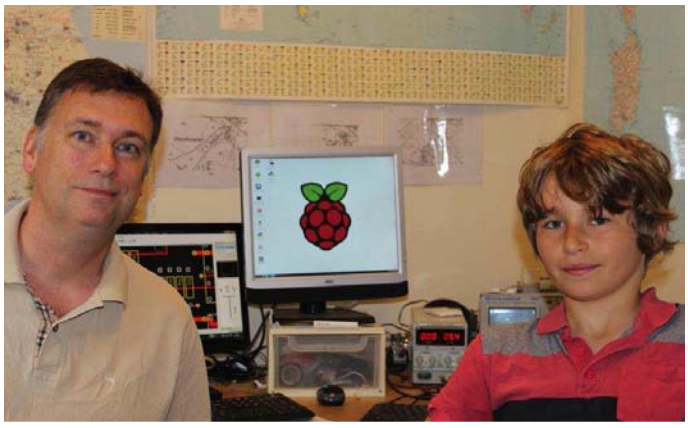
Prototype area with DC-DC converter and Real Time Clock

Developing a product

Our Raspberry Pi could now be run from a battery, keep the time and control things anywhere. We were mobile!

In fact it was working so well, and was so easy to use compared with other boards, we decided that other people might want to use it to. A friend of ours told me about Kickstarter and we decided to try and raise £1500 to build 100 units.

Before we could start this project we needed to come up with a name for the product. For a few weeks it was called the "Brabboard" but eventually we decided to call it the "Matboard".



Martin and Matthew in their home laboratory

The first PCB (version 1) had a few design problems due to some confusion on the relay manufacturers datasheet. We had connected the relay coil in reverse. The board didn't work!

It was time to introduce my son to electronics troubleshooting and re-work or, as he likes to call it, "hacking". Luckily it was only a double sided board and we were able to drill out some tracks and add some wires! It didn't look great but it worked, and we had a product.

However we had made a few mistakes. In addition to the relay issue, the PCB terminal block was too small and difficult to use. We also had short circuits that had to be removed and the Raspberry Pi mounting holes were too close to the relays to be useful. Lesson learned... always build a prototype!

We decided to design a version 2, to fix all of the issues, add a silkscreen layer with clear labels for each GPIO pin and also connect the switches differently so they could be activated by either grounding the input or by applying a voltage (up

to 30VDC). We ended up also routing these through the ULN2803A. This gave the board the ability to read sensors with output voltages down to 2VDC. We also moved to a 4 layer PCB with power and ground planes to improve the layout of the board.

The new PCB took 3 weeks to arrive, as we were using the lowest cost delivery. In fact we could have got it in two days but that would have cost over £400!

Launch on Kickstarter

With version 2 working and the design finished, it was time to launch on Kickstarter. You can see our project page at http://goo.gl/Hg1JVB.

Putting the Kickstarter project together was also a lot of fun, especially making the ninety second video with Matthew.

It was also incredible how quickly we started to get enquiries and backers. Our first backer for the project was after only ten minutes of being live! At the beginning we were worried that we would not get the funding, but after only four days we had over forty backers and were two-thirds of the way to the £1500 goal.

Along the way we had a lot of people make suggestions and send us messages of encouragement. One idea was that the GPIO connector would be more usable if it was on the other side of the board, otherwise when the Raspberry Pi is installed on top of the Matboard the ribbon cable would block the LEDs and switches. Matthew and I agreed and decided that we would incorporate this suggestion. We were on to version 3!

A last minute change!

After a hectic few days finishing the version 3 design, we were ready to order the final prototype PCB. At the last minute, I came up with the idea of adding a couple of extra PCB holes that could provide extra 0V, 5V and 3.3V sources. I did this quickly before work and then sent the files to the PCB manufacturer. It proved to be a big mistake - in my rush I had connected all of the power and ground planes together.

A week into the order, we were looking at the PCB design when we spotted the error... it was a horrible, sickening feeling. The PCB manufacturer was helpful but the PCB was already in manufacture. There was no choice but to fix it and order another board. The PCB company took pity on us and gave us a five day delivery service at the three week price. It was an expensive mistake but at least we were on track.

The Kickstarter project finished and we exceeded our funding goal by over £1000. 105 people (mostly strangers) had backed the project from all over the world.

Along comes the Model B+

Now it was time to deliver the finished product. We had promised our backers that we would ship the Matboard kits by the end of August 2014 and we had a lot to do when another hiccup happened just before we were about to order the components. The Raspberry Pi Foundation announced the new Raspberry Pi Model B+. The new Model B+ has a 40-way GPIO connector compared to a 26-way connector on the original Raspberry Pi. The people backing the project were presumably owners of Raspberry Pi's with 26-way connectors, so we needed a solution that would enable our backers to use the board and also be useful to future customers using the Model B+.

Fortunately the first 26 GPIO pins on the new Model B+ have exactly the same function as the original unit. So we decided to change the connector on the Matboard from a shrouded type to an open type, so that both 26-way and 40-way cables can be used.

The Matboard will now work with the Model A, Model B and Model B+ Raspberry Pi's.

Future plans

For the Matboard, we finished the first kits for our 105 backers at the end of September. We have also ordered some extra kits for us to sell online at our website http://www.wlabs.co.uk. We are also hoping to find some other interested companies to help us promote the Matboard.

We are always thinking about new products and things to do with the Raspberry Pi and electronics in general. At the moment we are working on an LED cube with 9 LEDs on each face that can be made to flash in sequence. Matthew is also working on his own project that is a bit like Skylanders™ which can have up to 2 players on the Raspberry Pi.

What we have learnt

Matthew: I have learnt about electronic components and what they are used for on the board. I have learnt several soldering skills and can build a finished Matboard in 30 minutes. I also know how to design a PCB and have learned some programming skills like how to use a button with the GPIO.

Martin: From what I have seen, the Raspberry Pi has done a great job of getting Matthew involved in electronics and programming. It is quite fascinating to see him come up with an idea, write the code and wire the Matboard into a project.

The Tomato Watering Project

When we are on holiday we needed a solution to water our tomatoes in the greenhouse. Asking our neighbour to do this seemed too simple(!), so we created the following setup.

The idea was to use water from a waterbutt that is near the greenhouse with a small 12V pump. Luckily our greenhouse is near the house, so we were able to pick up a strong Wi-Fi signal.

It uses a Python file that is executed at 18:01 every day using `crontab` on the Raspberry Pi. We then read a water level sensor that is mounted with LEGO® bricks to determine if there is sufficient water to start the pump cycle. If the water level is low then a function is called to send an alert email.

Raspberry Pi and Matboard controlling a water pump

If the water level is OK, an electronic valve opens and the pump runs for two minutes. This is connected to a plastic pipe with holes drilled in it, which works surprisingly well.

After two minutes the valve closes and the pump stops. Without the valve we found that the siphon effect kicks in and empties the whole water butt into the green house (something else I learnt the hard way).

The Python code executes a command to take a picture of the greenhouse and emails it from a Gmail account.

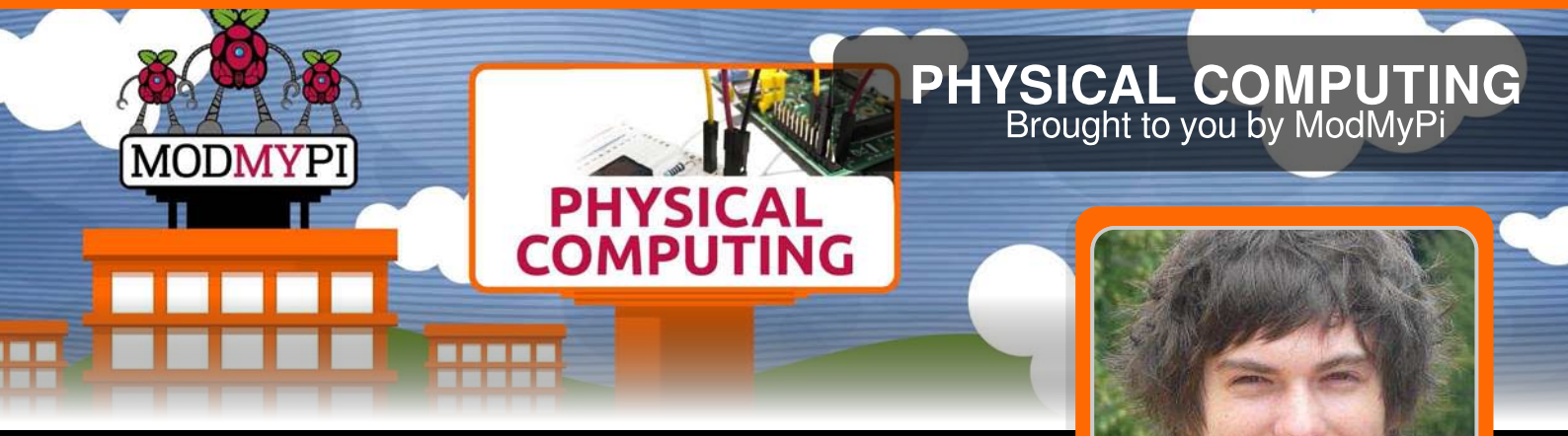
The system is powered by a 24W AC-DC PSU that delivers 12V to power the pump and valve. A 12V to 5V DC-DC converter is installed on the prototype area, to tap off 5V for the Raspberry Pi and Matboard.

We also used a DS1307+ Real Time Clock chip so that if we lose the Wi-Fi connection or have any kind of power outage, the system can still keep time and activate at 18:01.

We plan to add a few more features, that will use extra inputs and outputs of the Matboard over the next few months.

Python code

```
#!/usr/bin/python
#
# Note: you must change lines 29, 30 & 31

import RPi.GPIO as GPIO, feedparser
from time import sleep
import smtplib, os, sys
from email.mime.text import MIMEText
from email.MIMEMultipart import MIMEMulti
part
from email.MIMEImage import MIMEImage

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)

# Set up GPIO Inputs
# Yellow Input
GPIO.setup(7, GPIO.IN)
# Orange LED
GPIO.setup(12, GPIO.OUT)

#Set LEDs to Off and outputs to False
GPIO.output(12,False)

# Send email message and photo from Gmail
def send_email(msg):
  # Change these 3 lines to your details
  USERNAME = "me@gmail.com"
  PASSWORD = "your Gmail password"
  MAILTO = "someone@example.com"

  msg['From'] = USERNAME
  msg['To'] = MAILTO

  msg.attach(MIMEImage(file("/home/pi/ma
gpi/water.jpg").read()))

  server = smtplib.SMTP('smtp.gmail.com:5
87')
  server.ehlo_or_helo_if_needed()
  server.starttls()
  server.ehlo_or_helo_if_needed()
  server.login(USERNAME, PASSWORD)
  server.sendmail(USERNAME, MAILTO, msg.a
s_string())
  server.quit()

  print"Email sent to: " + MAILTO
  return

# Send email when there is no water
def Send_nowater_email():
  print"No water"
  msg = MIMEMultipart()
  msg.attach(MIMEText('Water Butt Empty')
)
  msg['Subject'] = 'Water Butt Empty'
  send_email(msg)
  return

# Send email when greenhouse is watered
def Send_watered_email():
  print"Sending image"
  msg = MIMEMultipart()
  msg.attach(MIMEText('Greenhouse watered
'))
  msg['Subject'] = 'Greenhouse Watered'
  send_email(msg)
  return

# Turn on the water pump
def water_plants():
  GPIO.output(12,True)
  sleep(120)
  GPIO.output(12,False)
  return

# Take a picture called water.jpg
def take_picture():
  os.system("raspistill -o /home/pi/magpi
/water.jpg -w 1024 -h 768 -q 30")

# Main control loop
while True:
  Input_yellow = GPIO.input(7)
  print Input_yellow

  if Input_yellow == False:
    water_plants()
    take_picture()
    Send_watered_email()

    print"Wait 30 seconds"
    sleep(30)
    print"Exit program"
    sys.exit()

  if Input_yellow == True:
    take_picture()
    Send_nowater_email()

    print"Wait 30 seconds"
    sleep(30)
    print"Exit program"
    sys.exit()
```



GPIO Sensing: HC-SR04 ultrasonic range finder - Part 3

Jacob Marsh

ModMyPi

SKILL LEVEL : BEGINNER

In previous tutorials we have outlined temperature sensing, PIR motion controllers plus buttons and switches, all of which can plug directly into the Raspberry Pi's GPIO ports. The HC-SR04 ultrasonic range finder is similarly very simple to use, however the signal it outputs needs to be converted from 5V to 3.3V so as not to damage the Raspberry Pi! We will introduce some physics along with electronics in this tutorial, in order to explain each step.

What you will need

HC-SR04
1kΩ resistor
2kΩ resistor
Jumper wires

Ultrasonic distance sensors

Sound consists of oscillating waves travelling through a medium (such as air) with the pitch being determined by the closeness of those waves to each other, defined as the frequency. Only some of the sound spectrum (the range of sound wave frequencies) is audible to the human ear, defined as the "acoustic" range. Very low frequency sound below acoustic is defined as "infrasound", with high frequency sounds above, called "ultrasound". Ultrasonic sensors are designed to sense object proximity or range using ultrasound reflection, similar to radar, to calculate the time it takes to reflect ultrasound waves between the sensor and a solid object. Ultrasound is mainly used because it is inaudible to the human ear and is relatively accurate within short distances. You could of course use acoustic sound for this purpose, but you would have a noisy robot, beeping every few seconds.

A basic ultrasonic sensor consists of one or more ultrasonic transmitters (basically speakers), a receiver and a control circuit. The transmitters emit a high frequency ultrasonic sound, which bounces off any nearby solid objects. Some of that ultrasonic noise is reflected and detected by the receiver on the sensor. That return signal is then processed by the control circuit to calculate the time difference between the signal being transmitted and received. This time can subsequently be used, along with some clever math, to calculate the distance between the sensor and the reflecting object.

The HC-SR04 Ultrasonic sensor we will be using in this tutorial for the Raspberry Pi has four pins: Ground (GND), Echo Pulse Output (ECHO),

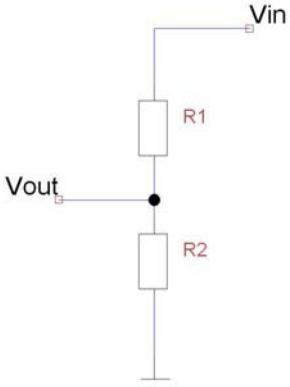
Trigger Pulse Input (TRIG), and 5V supply (Vcc). We power the module using Vcc, ground it using GND and use our Raspberry Pi to send an input signal to TRIG, which triggers the sensor to send an ultrasonic pulse. The pulse waves bounce off any nearby objects and some are reflected back to the sensor. The sensor detects these return waves and measures the time between the trigger and returned pulse, and then sends a 5V signal on the ECHO pin.

ECHO will be “low” (0V) until the sensor is triggered when it receives the echo pulse. Once a return pulse has been located ECHO is set “high” (5V) for the duration of that pulse. Pulse duration is the full time between the sensor outputting an ultrasonic pulse and the return pulse being detected by the sensor receiver. Our Python script must therefore measure the pulse duration and then calculate the distance.

NOTE: The sensor output signal (ECHO) on the HC-SR04 is rated at 5V. However, the input pin on the Raspberry Pi GPIO is rated at 3.3V. Sending a 5V signal into that unprotected 3.3V input port could damage your Raspberry Pi, which is something we want to avoid! We will need to use a small voltage divider circuit, consisting of two resistors, to lower the sensor output voltage to something our Raspberry Pi can handle.

Voltage dividers

A voltage divider consists of two resistors (R1 and R2) in series connected to an input voltage (Vin), which needs to be reduced to our output voltage (Vout). In our circuit, Vin will be ECHO, which needs to be decreased from 5V to our Vout of 3.3V.



The following circuit and simple equation can be applied to many applications where a voltage needs to be reduced. If you do not want to learn how this works, just get a 1kΩ resistor and a 2kΩ resistor.

$$Vout = Vin \times \frac{R2}{R1 + R2}$$

$$\frac{Vout}{Vin} = \frac{R2}{R1 + R2}$$

Without getting too deep into the math side, we only actually need to calculate one resistor value, as it is the dividing ratio that is important. As we know our input voltage (5V) and our required output voltage (3.3V), so we can use any combination of resistors to achieve the reduction. I decided to use a 1kΩ resistor in the circuit as R1. Adding the other values to the equation gives the following result:

$$\frac{3.3}{5} = \frac{R2}{1000 + R2}$$

$$0.66 = \frac{R2}{1000 + R2}$$

$$0.66(1000 + R2) = R2$$

$$660 + 0.66R2 = R2$$

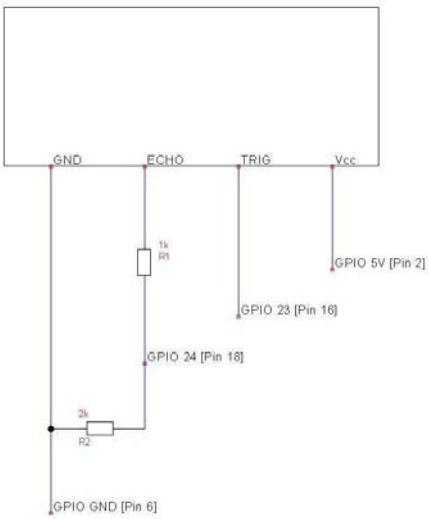
$$660 = 0.34R2$$

$$1941 = R2$$

So, we will use 1kΩ for R1 and a 2kΩ resistor for R2.

Assemble the circuit

We will use four pins on the Raspberry Pi for this project: GPIO 5V [Pin 2] → Vcc (5V Power), GPIO GND [Pin 6] → GND (0V Ground), GPIO 23 [Pin 16] → TRIG (GPIO Output) and GPIO 24 [Pin 18] → ECHO (GPIO Input).



1. Plug four of your male to female jumper wires into the pins on the HC-SR04 as follows: red → Vcc, blue → TRIG, yellow → ECHO and black → GND.

2. Plug Vcc into the positive rail of your breadboard and plug GND into your negative rail.

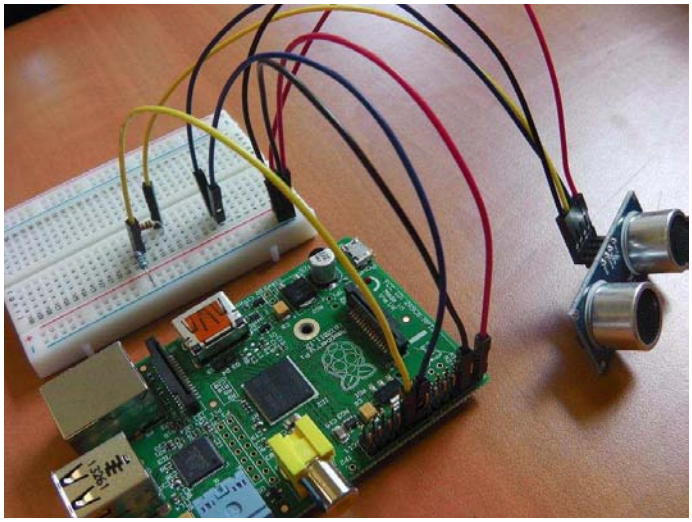
3. Plug GPIO 5V [Pin 2] into the positive rail and GPIO GND [Pin 6] into the negative rail.

4. Plug TRIG into a blank rail and plug that rail into GPIO 23 [Pin 16]. (You can plug TRIG directly into GPIO 23 if you want).

5. Plug ECHO into a blank rail and link another blank rail using R1 (1kΩ resistor).

6. Link your R1 rail with the GND rail using R2 (2kΩ resistor). Leave a space between the two resistors.

7. Add GPIO 24 [Pin 18] to the rail with your R1 (1kΩ resistor). This GPIO pin needs to sit between R1 and R2.



That's it! Our HC-SR04 sensor is connected to our Raspberry Pi.

Sensing with Python

Now that we have connected our ultrasonic sensor to our Raspberry Pi, we need to program a Python script to detect distance. The ultrasonic sensor output (ECHO) will always output low (0V) unless it has been triggered, in which case it will output 5V (3.3V with our voltage divider). We therefore need to set one GPIO pin as an output, to trigger the sensor, and one as an input to detect the ECHO voltage change.

First, import the Python GPIO library, import our time library (so we make our Raspberry Pi wait between steps) and set our GPIO pin numbering:

```
import RPi.GPIO as GPIO
import time
GPIO.setmode(GPIO.BCM)
```

Next, we name our output pin GPIO 23 (which triggers the sensor) as TRIG and our input pin GPIO 24 (which reads the return signal) as ECHO:

```
TRIG = 23
ECHO = 24
```

We then print a message to let the user know that distance measurement is in progress:

```
print "Distance Measurement In Progress"
```

Next, set the two GPIO ports as inputs and outputs as defined previously:

```
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)
```

Then, ensure that the TRIG pin is set low and give the sensor a couple of seconds to settle:

```
GPIO.output(TRIG, False)
print "Waiting For Sensor To Settle"
time.sleep(2)
```

The HC-SR04 sensor requires a short 10µS pulse to trigger the module, which will cause the sensor to start the ranging program (8 ultrasound bursts at 40 kHz) in order to obtain an echo response. So, to create our trigger pulse, we set the trigger pin high for 10µS then set it low again:

```
GPIO.output(TRIG, True)
time.sleep(0.00001)
GPIO.output(TRIG, False)
```

Now that we have sent our pulse signal we need to

listen to our input pin, which is connected to ECHO. The sensor sets ECHO to high for the amount of time it takes for the pulse to go and come back, so our code therefore needs to measure the amount of time that the ECHO pin stays high. We use a `while` loop to ensure that each signal timestamp is recorded in the correct order.

The `time.time()` function will record the latest timestamp for a given condition. For example, if a pin goes from low to high, and we are recording the low condition using the `time.time()` function, the recorded timestamp will be the latest time at which that pin was low.

Our first step must therefore be to record the last low timestamp for ECHO (`pulse_start`) i.e. just before the return signal is received and the pin goes high:

```
while GPIO.input(ECHO) == 0:
  pulse_start = time.time()
```

Once a signal is received, the value changes from low (0) to high (1) and the signal will remain high for the duration of the echo pulse. We therefore also need the last high timestamp for ECHO (`pulse_end`):

```
while GPIO.input(ECHO) == 1:
  pulse_end = time.time()
```

We can now calculate the difference between the two recorded timestamps and hence the duration of the pulse (`pulse_duration`):

```
pulse_duration = pulse_end - pulse_start
```

Knowing the time it takes for the signal to travel to an object and back again, we can now calculate the distance using the following formula:

$$Speed = \frac{Distance}{Time}$$

The speed of sound is variable, depending on what medium it is travelling through, in addition to the temperature of that medium. However, the speed of sound at sea level is 343m/s (or 34300cm/s) so we will use this as our baseline.

We also need to divide our time by two because what we have calculated above is actually the time it takes for the ultrasonic pulse to travel the distance to the object and back again. We simply want the distance to the object! We can simplify the calculation to be completed in our Python script as follows:

$$34300 = \frac{Distance}{Time/2}$$

$$17150 = \frac{Distance}{Time}$$

$$17150 \times Time = Distance$$

We can plug this calculation into our Python script:

```
distance = pulse_duration * 17150
```

Now we round our distance to 2 decimal places:

```
distance = round(distance, 2)
```

Then we print the distance. This command will print the word “Distance:” followed by the distance variable, followed by the unit “cm”:

```
print "Distance:", distance, "cm"
```

Finally, we clean our GPIO pins to ensure that all inputs and outputs are reset:

```
GPIO.cleanup()
```

Save your Python script as `range_sensor.py` and run it using the following command:

```
sudo python range_sensor.py
```

```
pi@raspberrypi ~ $ sudo python range_sensor.py
Distance Measurement In Progress
Waiting For Sensor To Settle
Distance: 12.52 cm
pi@raspberrypi ~ $
```

The MagPi What's On Guide

Want to keep up to date with all things Raspberry Pi in your area?

Then this section of The MagPi is for you! We aim to list Raspberry Jam events in your area, providing you with a Raspberry Pi calendar for the month ahead.

Are you in charge of running a Raspberry Pi event? Want to publicise it?
Email us at: editor@themagpi.com

SWAMP Fest (South WAles Makers & Programmers)

When: Saturday 11th October 2014, 10.00am to 5.00pm
Where: TechHub Swansea, 11 Wind Street, Swansea, SA1 1DP, UK

Meet the coder, hacker, maker and associated groups in and around South Wales. With a large number of talks and workshops covering Arduino, Minecraft, Qt5, 3D printing and more besides this event looks set to be great fun and highly interesting. http://www.eventbrite.co.uk/e/12677094531.

Come and meet The MagPi on our stand. We'll have demonstrations of HDMIPi and the new Model B+ as well as various other hardware boards. In the afternoon we'll be giving a short talk about the magazine and giving you the opportunity to ask us, well, anything you want really. We look forward to seeing you in Swansea.

Huddersfield Raspberry Jam

When: Saturday 25th October 2014, 10.00am to 3.00pm
Where: Huddersfield Library, Princess Alexandra Walk, Huddersfield, HD1 2SU, UK

A Halloween themed Raspberry Jam! Make something Halloween themed with your Raspberry Pi and bring it along. http://huddersfieldraspberryjam.co.uk/

London Raspberry Jam

When: 8th November 2014, 11.00am to 6.00pm
Where: Kano, 69-89 Mile End Road, London, E1 4TT, UK

A celebration of everything to do with DIY technology: soldering, 3D printing, robots, gaming, virtual reality and more! The event is open to all ages. http://www.eventbrite.co.uk/e/13310980501

Bristol Digimakers

When: Saturday 29th November 2014, 10.30am to 4.30pm
Where: At-Bristol, Anchor Road, Bristol, BS1 5DB, UK

Digimakers is a series of technology events aimed at children (7+), teachers and parents. With lots of workshops the event provides a great opportunity to learn about electronics and computing: from programming to hacking hardware. https://www.facebook.com/digimakersbristol

Build QR Code structures inside Minecraft

Dougie Lawson
MagPi Writer

SKILL LEVEL : BEGINNER

In this article I am going to show you how you can dynamically display QR codes inside Minecraft: Pi Edition. You can use these for many ideas - from displaying clues to puzzles in your Minecraft world to linking to websites. Readers are also invited to read our other Minecraft articles in Issue 11 and Issue 23.

Getting started

If you have the latest version of Raspbian then Minecraft already comes preinstalled. If you do not have the latest Raspbian then let's get going by installing Minecraft and testing that it works.

Open an LXTerminal window and enter the following commands:

```
cd ~
wget https://s3.amazonaws.com/assets.minecra
ft.net/pi/minecraft-pi-0.1.1.tar.gz
tar -zxvf minecraft-pi-0.1.1.tar.gz
```

Make a note of `/home/pi/mcpi/api/python` as we will need that later. Test Minecraft with the following commands:

```
cd ~/mcpi
./minecraft-pi
```

You should get the familiar screen for Minecraft and can build a new world. Have a play around and when you are done press <TAB> to get control of the cursor. We can switch to the LXTerminal window and close down Minecraft by pressing the <CTRL>+<C> keys together. We will come back to Minecraft later when we are ready to run the Python program that is going to build our structure in the virtual world.

Getting QR encoder and testing your first QR code

Although we could build our own QR code maker in Python, to keep things very simple and give us instant results we will use a ready built QR code generator called `qrencode`.

Open an LXTerminal and issue the following commands:

```
sudo apt-get install qrencode
qrencode -t ANSI "Hello World"
```

We have now got a QR code displaying in our LXTerminal window. You may need to stretch the window size so that the white borders at the top and bottom are visible. Get your smart phone and scan that QR code using any QR code app to prove that everything is working correctly.

The qrencode program can generate QR codes in various formats. In our Python program we are going to generate an ASCII (plain text) format QR code and edit the output with the stream editor, sed. The qrencode program generates "##" for every black square and a space for every white square in the QR code, when we use the -t ASCII option.

Open an LXTerminal and issue the following complex command:

```
qrencode -t ASCII "Hello Minecraft" | \
sed 's/  / /g' | sed 's/##/#/g' > ~/mc.qr.txt
```

Note the two spaces in 's/ / /g'. We have now generated the data that will be the input for our Python program. Take a look at it with the cat command or use the Leafpad editor.

Glueing it all together

To build a QR code structure in Minecraft we are going to use the Python application programming interface (API). That allows us to control our Minecraft world with a program. For example, we can discover Steve's co-ordinates (Steve is the protagonist, i.e. your playable character) and we can teleport Steve from one location to another. We can also add or delete blocks in the world.

So, we have the data built from qrencode and sed and we have a method to read that data and build some blocks in our Minecraft world. Let's look at the Python program to do this. The program is called mc.qr.py and I have stored it in a directory called python, inside the home directory.

First we import the sys package and add the API directory to the system path so that Python can find the Minecraft Python API packages. Note, please choose the correct path depending if you installed Minecraft or it came pre-installed:

```
#!/usr/bin/python
import sys
sys.path.insert(1,'/home/pi/mcpi/api/python')
#sys.path.insert(1,'/opt/minecraft-pi/api/
python')
```

Now we can import the API packages to connect to the Minecraft world and build blocks:

```
import mcpi.minecraft as mine
import mcpi.block as block
```

Next we create a connection and get the current co-ordinates where Steve is standing:

```
mc = mine.Minecraft.create()
pPos = mc.player.getTilePos()
print "Player point:", pPos.x, pPos.y, pPos.z
```

Now move Steve (teleport him) twenty blocks back, forty blocks down and twenty blocks to his left and read his new position:

```
mc.player.setTilePos(pPos.x - 20, pPos.y - 40,
pPos.z - 20)
nPos = mc.player.getTilePos()
```

Open the input file generated by qrencode and read every line into an array:

```
qrc = open('mc.qr.txt', 'r')
arrayQR = []
for line in qrc:
  arrayQR.append(line)
qrc.close()
```

We then initialise the variables we use to position the blocks as we generate them in the Minecraft world:

```
print "Starting point:",nPos.x,nPos.y,nPos.z
x = nPos.x
y = nPos.y
z = nPos.z
```

The file generated by `qrencode` would come out upside down if we just worked from the first record to the last. So we need to read the last line first (to build the left hand line of blocks) and work backwards to the top (right hand line of blocks). Python has the `reversed()` function to read arrays from the last record to the first record. So each time round this `FOR` loop, the variable `i` is the line we are working on.

We set the starting position for the height of the block above Steve's position to the length of each record and work down while reading the line from left to right. Trust me, that gets the QR code turned through 90 degrees but it is not back to front. QR codes have the property that you can read them up, down, left or right (those three large squares in the corners tell the QR code reader how the code is oriented).

As we read the file, if the character is a space (white square) we build a block of snow (block id = 80) in the world. If the character is a hash (black square) we build a block of obsidian (block id = 49). You can find the block id codes on the Minecraft Wiki at http://minecraft.gamepedia.com/Data_values/Block_IDs:

```
for i in reversed(arrayQR):
  y = pPos.y + len(i)

  for j in range(0, len(i)):
    if (i[j] == " "):
      block = 80
    if (i[j] == "#"):
      block = 49
    y = y - 1
    mc.setBlock(x, y, z, block)
  x = x + 1
```

Finally we teleport Steve back to his original position:

```
mc.player.setPos(pPos.x, pPos.y, pPos.z)
```

Display the QR code in Minecraft

To get everything running we need to start Minecraft then when it is running we start our Python program. So we need two LXTerminal windows. In the first LXTerminal window enter the following commands:

```
cd ~/mcpi
./minecraft-pi
```

Login to your Minecraft world and find an area without too much terrain (we don't want trees blocking our view). Then press the <ALT>+<TAB> keys together to switch to the second LXterminal window. In that second window enter the following commands:

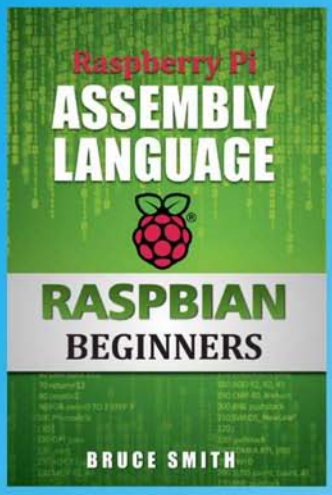
```
cd ~/python
chmod 755 ./mc.qr.py
./mc.qr.py
```

The screen will go blank because the last few blocks are built on top of Steve! When it is done drive round to find your QR code in the Minecraft world. I found I needed to build a tower of stone (or dirt) to get Steve positioned so that the QR code was square on and filled the top half of the screen without any keystoning effects. I was then able to scan it with my smart phone.

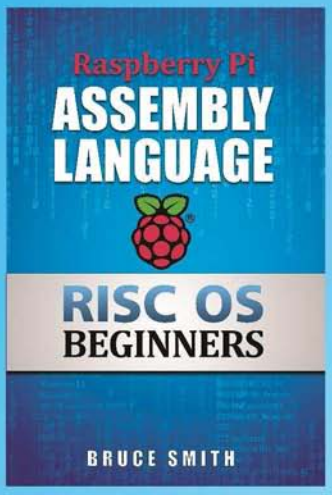
I'd like to thank Konrad, my 14 year old son and Minecraft expert, for his patience and advice. I started building the code with diamond (block id = 57) but Konrad suggested using snow for better contrast.

BOOKS FOR THE SERIOUS PROGRAMMER

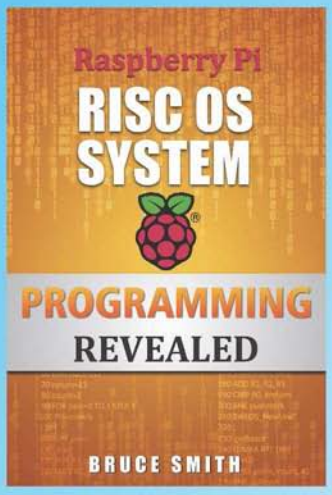
Full details at www.brucesmith.info Print and eBooks available.



With nothing other than Raspbian installed on your Raspberry Pi, this book shows you how to create your own machine code programs using ARM assembly language. With lucid descriptions, Bruce Smith keeps things simple and includes plenty of program examples you can try for yourself using the GCC Compiler.



Learn how to use the inbuilt BBC BASIC Assembler to create and generate machine code. Includes examples that show how to integrate the flexibility of BASIC into your assembler. Later chapters introduce the GCC Compiler and demonstrate how to use it's features to create stand alone machine code.



This 320-page book takes the lid off RISC OS. Aimed at those wishing to learn how to program RISC OS directly but are struggling with the Programmers Reference Manuals or simply don't know where to start. This book will teach you everything you need to know to get the most from RISC OS.

About Our Books

BSB books are ideal for the novice. Starting from first principles they lead you comfortably on your way to becoming an accomplished programmer. All books are fully supported at www.brucesmith.info with additional information, downloads, links, hints and tips.

Amazon 5-Star Reviews

All three books featured here have received 5-star reviews from readers on Amazon:

Excellent little book.

This is an excellent book which is a must-buy for anyone who wants to learn how to program their Raspberry Pi.

This is a great book for budding programmers.

Great book. An easy, step by step intro to assembly language.

Bruce has a very readable style of writing. The information is well presented.

"This is a great book. Bruce Smith writes great books."

Version control basics using Git - Part 1

SKILL LEVEL : BEGINNER

Alec Clews

Guest Writer

Introduction

When producing an article for The MagPi up to five different people may work on the article, in addition to the author. There is testing, layout, graphics, proof reading plus edits by the Issue Editor. To ensure that all changes are recorded and everyone is working on the latest version, we use a tool called Git. Git is widely used by many organisations so it is a useful skill to have. It is also a great tool for student projects. We asked Alec Clews to explain how it all works.

What is Version Control?

Version Control (VC) is a common practice used to track all the changes that occur to the files in a project over time. It needs a Version Control System (VCS) tool to work.

Think about how you work on a computer. You create stuff; it might be a computer program you are modifying, resume for a job application, a podcast or an essay. The process we all follow is usually the same. You create a basic version and you improve it over time by making lots of different changes. You might test your code, spell check your text, add in new content, re-structure the whole thing and so on. After you finish your project (and maybe release the content to a wider audience) the material you created can be used as the basis for a new project. A good example is writing computer programs, which usually consist of several different files that make up the project. Once you create a version you are happy with, programs often have to be changed many times to fix bugs or add new features. Programs are often worked on and modified by many different people, many of whom want to add features specific to their needs. Things can get confusing very quickly!

Because this article is written for users of the Raspberry Pi the examples we will use from now on will be based on software development projects, but remember that you can apply the principles to any set of computer files. *[Ed: For example the Scribus files we use at The MagPi.]*

How does a VCS work?

The way that a VCS works is by recording a history of changes. What does that mean?

Every time a change is completed (for example fixing a bug in a project) the developer decides that a logical "save" point has been reached and will store all the file modifications that make up the change in the VCS database.

Author photo courtesy of Jack Cotton

The term often used for a group of changes that belong together like this is a **changeset**. As well as changing lines of code in source files there might be changes to configuration files, documentation, graphic files and so on.

Along with the changes to the files the developer will be prompted by the VCS to provide a description of the change with a **commit message** which is appended to the **commit log**.

The process of storing the changes in the VCS database (usually referred to as the **repository** or repo for short) is called **making a commit**.

The hard work in making a commit is done by the VCS - all the developer does is issue the commit command and provide the commit message. The VCS software calculates which files have changed since the last commit and what has changed. It then stores these changes, plus the commit message, the date, time, name of the developer (committer) and other information in the repository.

Version Control is also sometimes referred to as Revision Control.

Now let us add another layer. Our project might be big enough that we are a team working on the project together and we all make changes to the digital files (also called **assets**). That will introduce a lot of potential problems. We will now talk about those and how a VCS can help.

Why is Version Control so important?

Imagine a software project. It might have hundreds of files (for example source code, build scripts, graphics, design documents, plans etc.) and dozens of people working on the project making different types of changes. There are several problems that will happen:

1. Two people might be editing the same file at once and changes can be overwritten.

2. After the project has been running for some time it is very hard to understand how the project has evolved and what changes have been made. How can we locate a problem that might have been introduced some time ago? Just fixing the problem may not be enough - we probably also need to to understand the change that introduced it.

3. If two people want to change the same file one will have to wait for the other to finish. This is inefficient.

4. If two people are making (long running) changes to the project it may take some time for both sets of changes to be compatible with each other. If the same copy of the project is being updated with both sets of changes then the project may not work correctly or even compile.

There are three core questions a VCS helps to answer via the commit history and commit messsage - what changes were made in the past, why were they made and who made them?

Individual developers find this information useful as part of their daily workflow and it also helps organisations with their compliance and audit management if needed.

There are also three core things a VCS helps do:

1. Undo a half complete or incorrect change made in error and roll back to a previous version.

2. Recreate a snapshot of the project as it was at some point in the past.

3. Allow two streams of changes to be made independently of each other and then integrate them at a later date (parallel development). This feature depends on the specific features of the VCS tool you are using.

You may find the article at http://tom.preston-werner.com/2009/05/19/the-git-parable.html useful in introducing important ideas.

Types of VCS tools available

Distributed vs Centralised

Modern VCS tools work on a distributed model (DVCS). This means that every member of the project team keeps a complete local copy of all the changes. The previous model, still widely used with tools like Subversion, is centralised. Here there is only one central database with all the changes and team members only have a copy of the change they are currently working on in their local workspace.

(In version control terminology a local workspace is often called a **working copy** and it will contain a specific revision of files plus changes.)

Open source and commercial tools

There are many commercial and open source tools available in the market. As well as the core version control operations, different tools will offer different combinations of features, support and integrations.

In this article we will be using a VCS called Git, a popular open source tool that uses a distributed model with excellent support for parallel development.

Summary

Version Control tools:

• Provide comprehensive historical information about the work done on a project.

• Help prevent the loss of information (e.g. edits being overwritten).

• Help the project team be more efficient by using parallel development (and often integrating with other tools such as bug tracking systems, project build systems, project management etc.)

• Help individual developers be more efficient with tools such as difference reports.

Example VCS operations using Git

The rest of this article will take a hands on approach by demonstrating the use of Git to manage a simple set of changes. You should follow along on your own Raspberry Pi using a new test project as explained below.

Git is a very popular DVCS originally developed to maintain the GNU/Linux kernel source code (the operating system that usually runs on the Raspberry Pi). It is now used by many very large open source projects and a lot of commercial development teams. Git is very flexible and thus has a reputation of being hard to use, but we are only going to concentrate on the ten or so commands you need to be useful day to day.

The following examples assume that you are using Raspbian Linux on a Raspberry Pi. First we are going to download an example Python project called Snakes, which we will store in a directory called `snakes`.

You can do that by running the following commands from the command line or in LXTerminal if you are using the desktop GUI:

```
cd ~
mkdir snakes
wget -O game.tar.gz http://goo.gl/nB4tYe
cd snakes
tar -xzf ../game.tar.gz
```

If you are unfamiliar with using commands from the terminal there is a tutorial on how to use the Linux command line at http://linuxcommand.org/learning_the_shell.php.

Git setup

Make sure you have the correct tools installed by typing the following commands:

```
sudo apt-get install git git-gui gitk
sudo apt-get install git-doc
```

Test the installation with the command:

```
git --version
```

You should see something like (or newer):

```
git version 1.17.10.4
```

Tell Git who you are. This is very important information and is recorded in every change you make. You must of course substitute your own name and email address in the correct places:

```
git config --global user.name "My Name"
git config --global user.email "a@b.com"
```

Git records that information in a user configuration file called `.gitconfig` in your home directory. Note that files and directories that are prefixed with a period (`.`) are hidden. If you enter the command `ls` you will not see these files. Instead enter `ls -A` to see everything.

In case you exchange files with developers working on a Microsoft Windows, (which is highly likely) you should also run the command:

```
git config --global core.autocrlf input
```

See https://help.github.com/articles/dealing-with-line-endings#platform-all for further details.

More information on setting up Git can be found at http://git-scm.com/book/en/Getting-Started-First-Time-Git-Setup.

Start a new project by creating a repo

The next thing we need to do is create an empty Git database called a repo (short for repository) inside our `snakes` directory. Enter:

```
cd snakes
git init
```

You should see something like:

```
Initialized empty Git repository in
/home/pi/snakes/.git/
```

Git has now created a hidden directory called `.git`. Remember, use `ls -A` to see it.

Next we issue a `git status` command. Notice that in Git all commands are typed after the word `git` (e.g. `git init` or `git status`). Enter:

```
git status
```

The output from the status command is:

```
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in
what will be committed)
#
#       game/
#       helloworld.py
#       if.py
#       maths.py
#       variables.py
#       while.py
nothing added to commit but untracked
files present (use "git add" to track)
```

We can ignore most of the detail for now. What is important is that Git:

1. Warns us that some files are not being controlled (untracked) by the VCS.

2. Lists the files and directories with their status. We will see this change as we progress further in the example.

Add the project files to Git

Before changes are added to the repo database we have to decide what will be in the commit. There might be many changes in the files we are working on, but our changset is actually only a small number of changes.

Git has a novel solution to this called the **index**. Before a file change can be committed to the repo it is first added to the index. As well as adding files to the index, files can also be moved or deleted. Once all the parts of the commit are complete, a `commit` command is issued.

The following examples are simple and for the time being you should just expect that before a commit is done changes are added to the index, as the following example shows. Note the trailing period (`.`) to represent the current directory and its subdirectories:

```
git add .
```

This command does not produce any output by default so do not be concerned if you get no messages. If you get a message similar to,

```
warning: CRLF will be replaced by LF
```

then this is normal as some versions of the Snakes project are provided in Windows format text files. You can fix this with the `dos2unix` utility.

If we run the `git status` command now we get different output:

```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to
unstage)
#
#       new file:   game/game0.py
#       new file:   game/game1.py
#       new file:   game/game2.py
#       new file:   game/game3.py
#       new file:   game/game4.py
#       new file:   game/snake.py
#       new file:   helloworld.py
#       new file:   if.py
#       new file:   maths.py
#       new file:   variables.py
#       new file:   while.py
```

This time each file that will be committed is listed, not just the directory, and the status has changed from `untracked` to new file.

Now that the file contents have been added to the index we can commit these changes as our first commit with the `git commit` command. Git adds the files and related information to our repo and provides a rather verbose set of messages about what it did. Enter:

```
git commit -m "Initial Commit"
```

The output from the command should be like:

```
[master (root-commit) 841ae8c] Initial
Commit
 11 files changed, 693 insertions(+)
 create mode 100755 game/game0.py
 create mode 100755 game/game1.py
 create mode 100755 game/game2.py
 create mode 100755 game/game3.py
 create mode 100755 game/game4.py
 create mode 100755 game/snake.py
 create mode 100755 helloworld.py
 create mode 100755 if.py
 create mode 100755 maths.py
 create mode 100755 variables.py
 create mode 100755 while.py
```

Now try the `git status` command again. The output is:

```
# On branch master
nothing to commit (working directory
clean)
```

This means that the contents of our working copy are identical to the latest version stored in our repo.

Another command worth running is `git log`, which is currently very brief as we have only have one commit. Mine looks like this:

```
commit 841ae8c672abac0ad9d8483fc3d68f060d9
dd5d8
Author: Pi <acdsip61-pi@yahoo.com>
Date:   Thu Aug 14 09:36:43 2014 +1000
    Initial Commit
```

The meaning of the Author, Date and comment field should be obvious. The commit field will be explained later. We now have our project under version control.

Coming up...

Next time we will see what happens when we make some changes.

Wyliodrin

Just drag & drop blocks to create your applications, using Visual Programming

Program and monitor your Pi from anywhere in the Internet

Use our graphs to display your sensors' data

Use a browser from any device to program and monitor your Raspberry Pi

For advanced users Wyliodrin has multiple programming languages and shell access

Java · python · C++ · ARDUINO · php · Perl · node.js

www.wyliodrin.com

Part 3: Keyboard input, animation and arrays

Jon Silvera

Guest Writer

SKILL LEVEL : BEGINNER

Hello once again and welcome to our FUZE BASIC tutorial. To further our exploration into the exciting world of BASIC we are going to get things moving. Over the next few pages we will add moving enemies (big rocks actually), plus the ability to fire. Of course firing a bullet is one thing but what happens when it hits something?

INKEY and scanKeyboard

Last month we added our player ship graphic and some movement commands to get us started. The movement commands need to be explained as there is some very useful stuff going on.

Generally the method for checking if a key has been pressed is to use the `INKEY` command as this will return the key being pressed when you check. For example `Key=INKEY` will either store the value -1 in the variable `Key` if no key is pressed, or the ASCII value of the key that is being pressed. The <Spacebar> has an ASCII value of 32, the <A> key is 65, the <a> is 97 and so on. You can write a simple program to help you determine the `INKEY` values.

Before we get started though please go back to the FUZE BASIC environment using the icon on the Desktop. Then go to the Editor <F2> and enter the following simple program:

```
CYCLE
PRINT INKEY
REPEAT
```

RUN <F3> this and hold down different keys to see what value they give you. Notice however that if you hold down two keys simultaneously only one of the numbers will be displayed. This means we cannot read two or more keys being pressed at the same time. This is not good for games programming so we added a command to FUZE BASIC to do exactly that.

To demonstrate this let's first get back to business. Press <ESC> to exit the program and then press <F2> twice to return to direct mode so we can load our program. Enter:

```
DIR
```

You should see a directory called `MagPi`. Enter:

```
CD MagPi
LOAD MagPi
```

Press <F2> to go to the Editor. Hopefully your program will be as we left it but if not you might want to go through and make sure all is as it should be.

The full program listing is as follows:

```
// MagPi Game
PROC Setup
  CYCLE
    PROC CheckControls
    PROC ScreenUpdate
  REPEAT
END

DEF PROC CheckControls
  UpKey = scanKeyboard (scanUp)
  DownKey = scanKeyboard (scanDown)
  LeftKey = scanKeyboard (scanLeft)
  RightKey = scanKeyboard (scanRight)
  IF UpKey THEN ShipY = ShipY + 1
  IF DownKey THEN ShipY = ShipY - 1
  IF LeftKey THEN ShipX = ShipX - 1
  IF RightKey THEN ShipX = ShipX + 1
ENDPROC

DEF PROC ScreenUpdate
  plotSprite (Ship, ShipX, ShipY, 0)
  UPDATE
ENDPROC

DEF PROC Setup
  HGR
  updateMode = 0
  ShipX = 0
  ShipY = gHeight / 2
  Ship = newSprite (1)
  loadSprite ("Player2.bmp", Ship, 0)
  setSpriteTrans (Ship, 255, 0, 255)
ENDPROC
```

If you RUN <F3> the program you can see we can use the cursor keys to move the ship around the screen. This is handled using the scanKeyboard (scanxxxx) command. The statement `RightKey = scanKeyboard (scanRight)` checks to see if the Right cursor key is being pressed. If it is it stores a 1 in the variable `RightKey`, or a 0 if it has not been pressed. This can be applied to every key so it's easy to check for more than one key press with a simple IF, AND, THEN statement. Press the <ESC> key and then <F2> to return to the editor.

gHeight and getSpriteH

We need to add a few restrictions so the player cannot zoom off the screen. At the same time we are going to add a simple animation to the player's ship. First, modify the `DEF PROC CheckControls` section so it is as follows:

```
DEF PROC CheckControls
  ShipID = 1
  UpKey = scanKeyboard (scanUp)
  DownKey = scanKeyboard (scanDown)
  LeftKey = scanKeyboard (scanLeft)
  RightKey = scanKeyboard (scanRight)
  IF UpKey AND ShipY <= (gHeight -
   getSpriteH (Ship)) THEN
    ShipY = ShipY + 4
    ShipID = 2
  ENDIF
  IF DownKey AND ShipY >= 0 THEN
    ShipY = ShipY - 4
    ShipID = 0
  ENDIF
  IF LeftKey AND ShipX >= 0 THEN ShipX =
ShipX - 4
  IF RightKey AND ShipX <= gWidth/2 THEN
ShipX = ShipX + 2
ENDPROC
```

Do not RUN it at this point as you will just get an error.

The `IF` statements check to see if the `ShipX` or `ShipY` positions are off the screen and if so they will not allow further movement. The check `IF UpKey AND ShipY <= (gHeight - getSpriteH (Ship))` looks much more complicated than it is. `gHeight` is a system variable that contains the maximum height of the current screen mode and `getSpriteH` checks the height of the sprite specified. We do not want the sprite to go any lower than the bottom of the screen, but this must take the height of the sprite into consideration. Try removing this part of the check to see what happens.

Adding animation

The `ShipID` variable is going to contain the player sprite ID so we can display different sprites just by changing this setting. You can see from the above the default setting is 1 and then depending if the Down or Up keys are pressed it can be 0 or 2 respectively. So, we need one graphic for Up, one for Down and a default one for when neither Up nor Down is pressed.

For the sake of gameplay we have also increased the speed dramatically as the ship now moves 4 pixels at a time, unless it is moving

forward where it has to work a bit harder and move forward at only 2 pixels at a time.

Because we have introduced a new variable, we need to change other parts too. Add this DrawShip procedure immediately after the END statement:

```
END

DEF PROC DrawShip
  plotSprite (Ship, ShipX, ShipY, ShipID)
ENDPROC
```

As we are about to have a lot more going on, we will be using separate functions for each of the main sprites. This one specifically draws the player's ship at the X and Y position. Change the beginning of the program to:

```
// MagPi Game
PROC Setup
  CYCLE
    PROC CheckControls
    PROC DrawShip
    UPDATE
  REPEAT
END
```

The UPDATE statement is now issued from the main loop. This will keep things running smoothly as everything will be updated at once. Because of this we no longer need the ScreenUpdate procedure so this can be deleted. Specifically, delete the following lines:

```
~~DEF PROC ScreenUpdate~~
~~  plotSprite (Ship, ShipX, ShipY, 0)~~
~~  UPDATE~~
~~ENDPROC~~
```

Nearly there. To introduce a new variable and add the extra ship graphics update the Setup procedure so it is the same as the following:

```
DEF PROC Setup
  HGR
  updateMode = 0
  ShipX = 0
  ShipY = gHeight / 2
  Ship = newSprite (3)
  loadSprite ("Player1.bmp",  Ship, 0)
  loadSprite ("Player2.bmp",  Ship, 1)
  loadSprite ("Player3.bmp",  Ship, 2)
  setSpriteTrans (Ship, 255, 0, 255)
  ShipID = 0
ENDPROC
```

As you can see we have added two extra sprites. The clever part is that only one main sprite container is required to hold all the sprites for that graphic. We use an index to determine which one to display. The 0, 1 and 2 at the end of each loadSprite command determines the index value. Then when we want to display the sprite we use plotSprite(name, x, y, index). You can have many sprites indexed so complex animations can be achieved by simply changing the index.

Ok, now you can RUN <F3> the program. If you get any errors you will need to go back and debug. Just make sure everything matches the listings provided and it should be fine. Now when you move around with the cursor keys, things should be much faster and best of all the ship will change depending if it is going up or down. It is a very simple but effective technique.

Adding enemies

It's time to add some scary monsters. Actually, monsters would take a bit more space to cover in this article so, we are going with rocks... but they are big, ugly ones if that helps!

Unfortunately there is no way to avoid this next part. We need to add a large section to the Setup procedure to introduce our enemies. We could have kept things very simple and just gone for one enemy sliding across the screen at a time, but where is the fun in that? We are going for waves of sixteen at a time (split into two lots of eight) at different speeds and with changing flying patterns.

The price to pay is in the typing so, heads down and get on with it. Add the following after the line ShipID = 0 in the Setup procedure:

```
  ShipID = 0
  EnemyMax = 63
  eID = 0
  EnemyID = 0
  EnemyX = 0
  EnemyY = 0
  EnemyActive = 1
  EnemyVariation = 0
  EnemyScore = 50
  EnemySpeed = 0
  DIM Enemy(EnemyMax, 6)
  DIM Rock(EnemyMax)

  FOR num = 0 TO EnemyMax CYCLE
    Rock(num) = newSprite (1)
    loadSprite("BigRock.bmp",Rock(num),0)
    setSpriteTrans(Rock(num),255,0,255)
  REPEAT

  EnemyCount = 0
  UNTIL EnemyCount > EnemyMax CYCLE
    READ EnemyX
    READ EnemyY
    READ EnemyVariation
    READ EnemyScore
    READ EnemySpeed
    EnemyScore = EnemyScore * EnemySpeed
    DATA 1280, 100, 3, 50, 2
    DATA 1280, 500, -3, 50, 2
    DATA 4000, 366, 4, 50, 3
    DATA 4000, 230, -4, 50, 3
    DATA 6000, 100, 6, 50, 3
    DATA 6000, 500, -6, 50, 3
    DATA 11000, 400, 5, 50, 4
    DATA 11000, 300, -5, 50, 4

    FOR num = 0 TO 7 CYCLE
      Enemy(EnemyCount + num, 0) =
Rock(EnemyCount + num)
      Enemy(EnemyCount + num, 1) = EnemyX
+ num * getSpriteW (Rock(0))
      Enemy(EnemyCount + num, 2) = EnemyY
      Enemy(EnemyCount + num, 3) =
EnemyActive
      Enemy(EnemyCount + num, 4) =
EnemyVariation
      Enemy(EnemyCount + num, 5) =
EnemyScore
      Enemy(EnemyCount + num, 6) =
EnemySpeed
    REPEAT

    EnemyCount = EnemyCount + 8
  REPEAT
ENDPROC
```

PROC Setup explanation

Let me explain what the lines we have added do. The first few lines simply introduce lots of new variables.

```
DIM Enemy(EnemyMax, 6)
DIM Rock(EnemyMax)
```

Dimension (DIM) variables are a very powerful type of variable called an array. It allows us to store multiple pieces of information in an index rather than just a single number in a single variable. Notice that Rock is a single dimension array and Enemy is a two-dimension array. We will talk more about arrays later.

```
FOR num = 0 TO EnemyMax CYCLE
```

The FOR loop is used to fill the Rock() array with sprite IDs, using the newSprite command, for each of the 64 rocks.

```
UNTIL EnemyCount > EnemyMax CYCLE
```

The UNTIL loop is used to fill the Enemy() array with the X and Y coordinates, the pattern used, the score value and the speed of each rock.

We are using the READ and DATA commands to easily add a whole load of information in one go. The main loop reads each of the lines of data and stores them in individual variables.

The enemy score is based on 50 * the enemy speed. However we are not including that this month so you will have to wait until next time.

```
FOR num = 0 TO 7 CYCLE
```

The rocks are configured in waves of eight. This smaller FOR loop is used to fill the Enemy() array with information in blocks of 8 at a time.

Again, do not RUN yet as it will not work without a few more changes.

Arrays

The array variable is very similar to using a database where you have a record and then various pieces of information are stored with that

record. We have a record ID, (e.g. 1) and then we store information within this like name, address, post code and so on. In the case of an array variable we can then refer to any part of this database with a simple index command.

Looking at the picture below can you tell me the value stored in our Enemy array at position 3, 4? Of course you can, it's -4.

	0 ID #	1 X Pos	2 Y Pos	3 Active	4 Pattern	5 Score	6 Speed
Enemy 0	0	1280	100	1	3	50	2
Enemy 1	1	1280	500	1	-3	50	3
Enemy 2	2	4000	366	1	4	50	3
Enemy 3	3	4000	230	1	-4	50	4
Enemy 4	4	6000	100	1	6	50	4

Using arrays allows us to store massive amounts of information all instantly accessible with a simple index command. For example Speed = Enemy(4, 6) should now make sense. Notice, as is usual with computer programming, that everything starts counting from zero.

Back to our program. Once again if we RUN at this point nothing will happen as we still need to add a few more bits elsewhere.

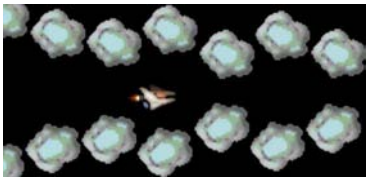
First we add a call to a new procedure called DrawEnemy in the main loop:

```
// MagPi Game
PROC Setup
  CYCLE
    PROC CheckControls
    PROC DrawShip
    PROC DrawEnemy
    UPDATE
  REPEAT
END
```

The following code is the actual procedure. Once again add this right after the END statement:

```
DEF PROC DrawEnemy
  FOR eID = 0 TO EnemyMax CYCLE
    IF Enemy(eID, 3) THEN
      Enemy(eID, 1) = Enemy(eID, 1) -
Enemy(eID, 6)
      EY = Enemy(eID, 2) + COS
(Enemy(eID, 1)) * Enemy(eID, 4) * 10
      plotSprite (Enemy(eID, 0),
Enemy(eID, 1), EY, 0)
    ENDIF
  REPEAT
ENDPROC
```

RUN <F3> the program to see how things are looking. All going well you should now have wave after wave of very scary rocks flying across the screen. Press <ESC> to stop the program when you have seen enough.



PROC DrawEnemy explanation

The PROC DrawEnemy section looks really complicated but when broken down it is, as always, easier than is first apparent.

```
FOR eID = 0 TO EnemyMax CYCLE
```

When the procedure is called it sets up a FOR LOOP to go through all 64 enemy sprites (EnemyMax is set to 63 and as we start counting from 0 this makes 64 in total).

```
IF Enemy(eID,3) THEN
  Enemy(eID,1)=Enemy(eID,1)-Enemy(eID,6)
```

The IF statement checks if the enemy is active or not. If it is then we reduce the X position Enemy(eID,1) by the speed stored in Enemy(eID,6).

```
EY = Enemy(eID,2) + COS (Enemy(eID,1)) *
Enemy(eID,4) * 10
```

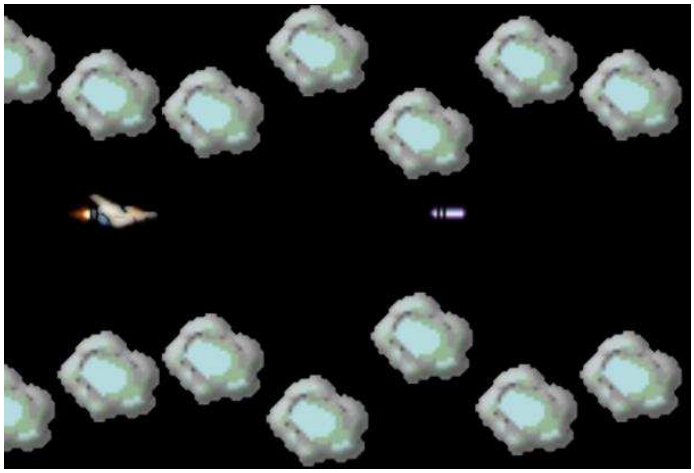
I don't like this bit as it makes me sound clever, and I'm not, but here goes. We are working out the Y position using the cosine of the X position (remember sine waves from Physics lessons). This causes the Y position to go up and down. Enemy(eID,4) gives us a variable to adjust the strength of the wave (the height).

```
plotSprite (Enemy(eID,0), Enemy(eID,1),
EY, 0)
```

As the X position is always moving to the left, a simple wave motion is formed. If you actually are clever then you can work out ways to make very complex patterns. Simple waves are about as much as I can manage!

Adding fire power

Just one last thing to do this month and that is to add the promised fire power. We are going to keep it simple as, once again, it would take too much space to do anything really fancy... but this will give you something to work with.



Add a call to a new procedure called `DrawBullet` in the main loop:

```
// MagPi Game
PROC Setup
  CYCLE
    PROC CheckControls
    PROC DrawShip
    PROC DrawEnemy
    PROC DrawBullet
    UPDATE
  REPEAT
END
```

In the `CheckControls` procedure we also add new code and a call to procedure `Bullet` to fire bullets:

```
  RightKey = scanKeyboard (scanRight)
  SpaceKey = scanKeyboard (scanSpace)

  IF SpaceKey AND NOT Fire THEN
    PROC Bullet
  IF UpKey AND ShipY <= (gHeight -
getSpriteH (Ship)) THEN
    ShipY = ShipY + 4
    ShipID = 2
  ENDIF
```

Add the following code to the end of the `DEF PROC Setup` procedure:

```
    EnemyCount = EnemyCount + 8
  REPEAT

  DIM Shot(3)
  Shot(0) = newSprite (1)
  loadSprite ("Bullet.bmp", Shot(0), 0)
  setSpriteTrans (Shot(0), 255, 0, 255)
  Fire = 0
ENDPROC
```

Finally add the `DrawBullet` and `Bullet` procedures directly below the END statement. These two procedures display the bullet and work out if it has hit anything with the `spriteCollidePP (Shot(0), 2)` command. More on this in a minute, but for now get busy and enter the following after the END statement:

```
DEF PROC DrawBullet
  IF Shot(1) > gWidth THEN
    hideSprite (Shot(0))
    Shot(3) = 0
    Fire = 0
  ENDIF
  IF Shot(3) THEN
    Shot(1) = Shot(1) + 6
    plotSprite (Shot(0), Shot(1),
Shot(2), 0)
    Hit = spriteCollidePP (Shot(0), 2)
    IF Hit > 0 AND Hit <= 64 THEN
      Enemy(Hit - 1, 3) = 0
      hideSprite (Hit)
      hideSprite (Shot(0))
      Shot(3) = 0
      Fire = 0
    ENDIF
  ENDIF
ENDPROC

DEF PROC Bullet
  Fire = 1
  Shot(1) = ShipX + getSpriteW (Ship) + 8
  Shot(2) = ShipY + getSpriteH (Ship) / 2
- 10
  Shot(3) = 1
ENDPROC
```

The `Bullet` procedure is very straightforward as it simply works out where the bullet should appear based on the current position of the player's ship. The `DrawBullet` procedure has a lot more going on and introduces several new sprite commands.

First, if the bullet goes off the screen then we set it to inactive and hide the sprite with the `hideSprite(Shot(0))` command.

`Hit = spriteCollidePP(Shot(0),2)` checks the sprite to see if it is in contact with any other sprite. If it is then it stores that sprite ID into the variable `Hit`. We then check to see if the sprite is a Rock. If so, we hide both the rock and the bullet and set them both to inactive so we don't draw them again elsewhere. We also reset the `Fire` variable so we can shoot again.

There are two kinds of collision detection - `spriteCollidePP(ID, accuracy)` and `spriteCollide(ID)`. The `PP` stands for "pixel perfect" so very accurate collisions can be checked. The standard version just checks the sprite's bounding box.

Coming up...

That's all we have got space for this month, actually I might be in trouble for taking up so much already!

Next issue will see the final part in this series. I hope to wrap things up with a few more collisions, scoring plus Start and Game Over scenes.

If you have not already noticed, in the next issue we are, in association with the very nice people at The MagPi, running a competition for the best game entry submitted using FUZE BASIC on a Raspberry Pi. Remember, you can download the FUZE BASIC boot image and Programmer's Reference Guide for free from the Resources page at http://www.fuze.co.uk.

FUZE BASIC

FUZE®

COMPETITION

TEASER

In the next issue, the folks at FUZE are planning to run a FUZE BASIC programming competition, with an incredible **£500** of prizes!

First prize is the amazing FUZE T2-R kit, worth £230. Not only does this have everything you need to maximise your enjoyment of the Raspberry Pi, it also includes an OWI programmable robotic arm kit!

Second prize is the superb FUZE T2-A kit, worth £180. Again, this contains everything you need including a Raspberry Pi Model B, solderless breadboard, various electronic components, SD card with FUZE BASIC, printed Programmer's Reference Guide and much more!

Third prize is the excellent FUZE T2-C kit for your Raspberry Pi. Worth £80, this kit contains the FUZE case, keyboard, integrated USB hub, FUZE I/O board and power supply.

Details of the prizes can be found at http://www.fuze.co.uk/products.

In this series you will learn everything that you need, but if you want to give yourself a head start you can download the FUZE BASIC Programmer's Reference Guide from http://www.fuze.co.uk/resources-2/.

FUZE®

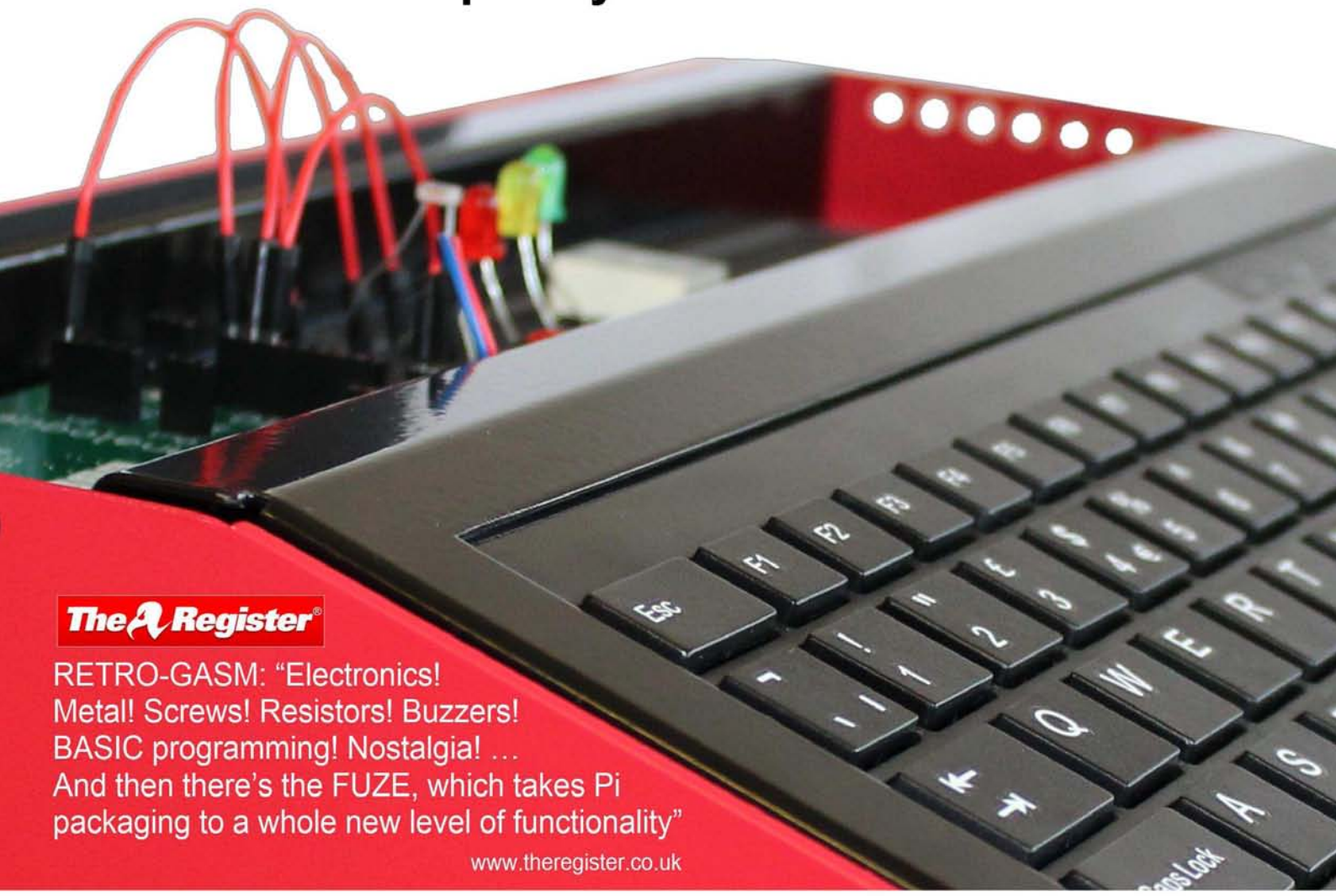
"The FUZE is what the Raspberry Pi was designed for"

THE NATIONAL MUSEUM OF COMPUTING

FUZE Type II

The Ultimate Case

for Raspberry Pi B & the new B+



The Register®

RETRO-GASM: "Electronics! Metal! Screws! Resistors! Buzzers! BASIC programming! Nostalgia! … And then there's the FUZE, which takes Pi packaging to a whole new level of functionality"

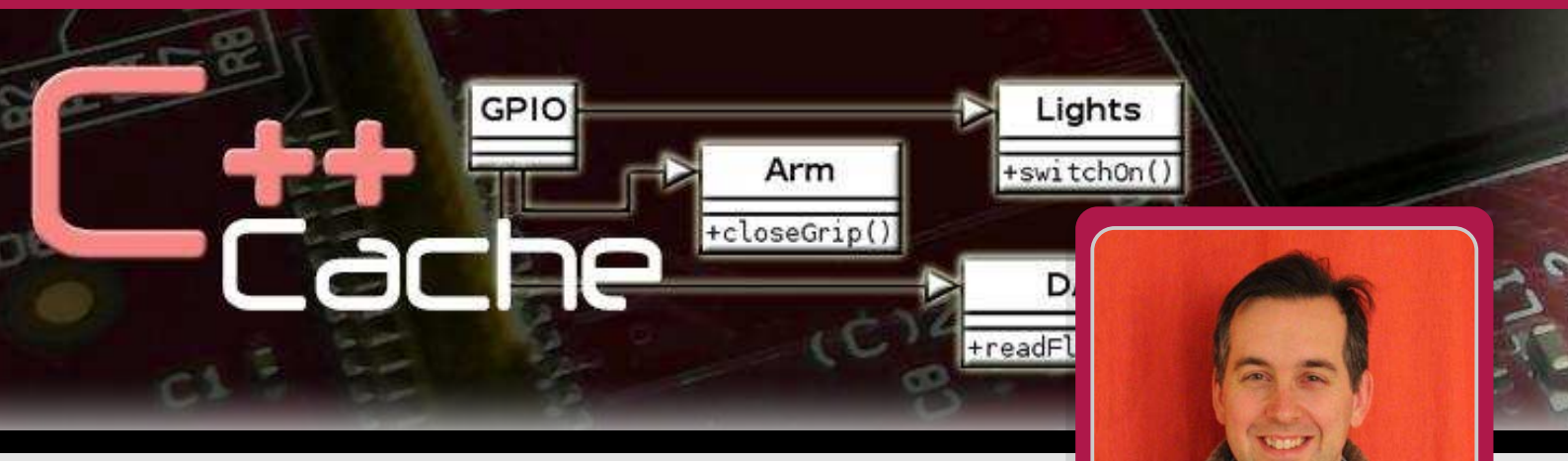
www.theregister.co.uk

Prices start from £89.99
See FUZE website for details

- Protect your Pi from physical & static damage
- UK keyboard* & 4 Extra USB ports
- FUZE I/O Board with GPIO pass-thru
- Clearly labeled input output ports
- 2 Amp power supply and on/off switch!
- Adds analogue ports, 4 in & 1 out

FUZE Technology Ltd - www.fuze.co.uk
+44 (0) 1844 239 432 - contact@fuze.co.uk

* USA & German keyboard layouts are also available



7 - Operator overloading

W. H. Bell

MagPi Writer

SKILL LEVEL : ADVANCED

The values contained in simple variables, such as `int` or `float`, can be added together using the mathematical operators that were introduced in the C Cave article in Issue 4 of The MagPi. This functionality can be extended to objects by the principle of operator overloading. Before continuing, it may be helpful to read through the introduction to C++ classes in Issues 23 and 24 of The MagPi.

Operators come in many shapes and sizes. There are mathematical operators, binary operators, relational operators, pointer syntax, stream operators, etc.. Each of these can be implemented as a function that deals with objects of a particular class. Since an operator that deals with objects is a function, the function could perform complicated operations to load data from disk or over the network before returning the result. Hopefully, the author of the C++ class has written sensible operator functions or provided documentation.

This tutorial introduces two simple mathematical and stream operator functions. The tutorial assumes that `g++` and `make` have been installed. Using the latest Raspbian image, these can be installed by typing:

```
sudo apt-get install -y g++ make
```

Two-dimensional vector

The power of operator overloading can be demonstrated with a simple example of numerical operators. In some mathematical problems one might have to use a two-dimensional vector, which has x and y components. From Issues 23 and 24, it is clear that a class can be written that contains x and y components as data members of a class. However, one would ideally like to be able to add vectors together or subtract them in a straight forward manner. This can be achieved by writing functions for the operators + and -. Create a new file called `TwoVector.h` and add the source code at the top of the next page. Then save the file.

```
#ifndef TWOVECTOR_H
#define TWOVECTOR_H

class TwoVector {
public:
  TwoVector(double x = 0., double y = 0.); // Constructor with default values
  double resultant(void) const; // Resultant
  double angle(void) const; // Angle of vector in x-y plane
  void rotate(double theta); // Rotate the two vector about itself
  TwoVector operator+(const TwoVector& twoVector) const; // Addition
  TwoVector operator-(const TwoVector& twoVector) const; // Subtraction
  TwoVector& operator=(const TwoVector& twoVector); // Assignment

  double x(void) const { return m_x; } // Return the x component
  double y(void) const { return m_y; } // Return the y component

private:
  double m_x; // x component of the vector
  double m_y; // y component of the vector
};

#endif
```

This class declaration includes three operator functions that operate on objects: to add, subtract and assign values. The class declaration also contains a constructor. The default values given in the constructor declaration are used if parameters are omitted when the constructor is called. There is a function to return the resultant of the vector, a function to return the angle of the vector in the x-y plane, a function to allow the vector to be rotated about itself and two functions that return the values of x and y components respectively. The class also contains two `private` data members that are present to store the values of the x and y components of the vector.

The `const` keyword is carefully used, to allow appropriate usage of the objects created. For example, functions that do not change values stored in the data members can safely be `const`. The `const` member functions are indicated by the keyword `const`, which is present just before the semicolon in each `const` function definition. These functions can be called from a `const` object of `TwoVector` type or from a normal object of `TwoVector` type. The operator functions use `const` references as parameters to avoid unnecesary copy constructors and to indicate that the operator will not change this parameter.

Now that the header file has been created, the implementation of the other member functions of the `TwoVector` class is needed. Therefore, create a new file called `TwoVector.cpp` and add:

```
#include "TwoVector.h"
#include <cmath>

TwoVector::TwoVector(double x, double y):
  m_x(x),  // assign the value of x to m_x
  m_y(y) { // assign the value of y to m_y
}

double TwoVector::resultant(void) const {
  double r = std::pow(m_x,2) + std::pow(m_y,2); // The sum of the squares of m_x and m_y
  if(r > 0.) r = std::sqrt(r);  // The sqrt only makes sense for values greater than zero
  return r;
}
```

```
double TwoVector::angle(void) const {
  double r = resultant();  // Get the resultant
  if(r <= 0.) return 0.;
  return std::acos(m_x/r); // angle in radians
}

void TwoVector::rotate(double theta) {
  double x = m_x, y = m_y; // Store the current values;
  m_x = x*std::cos(theta) - y*std::sin(theta);  // Rotate the x component
  m_y = x*std::sin(theta) + y*std::cos(theta);  // Rotate the y component
}

TwoVector TwoVector::operator+(const TwoVector& rhs) const {
  TwoVector twoVector = *this; // Copy this object using the assignment operator
  twoVector.m_x += rhs.m_x;  // Add the m_x value in this object to the other object
  twoVector.m_y += rhs.m_y;  // Add the m_y value in this object to the other object
  return twoVector;  // Return the resulting vector
}

TwoVector TwoVector::operator-(const TwoVector& rhs) const {
  TwoVector twoVector = *this; // Copy this object
  twoVector.m_x -= rhs.m_x;  // Subtract the m_x value from the other object from this class
  twoVector.m_y -= rhs.m_y;  // Subtract the m_y value from the other object from this class
  return twoVector;  // Return the resulting vector
}

TwoVector& TwoVector::operator=(const TwoVector& rhs) {
  m_x = rhs.m_x;  // Assign the m_x value from the other object
  m_y = rhs.m_y;  // Assign the m_y value from the other object
  return *this;  // Return the value of this object
}
```

When a member function or data member is private within a class definition, then it can be accessed directly by any objects instantiated from the class but cannot be accessed from objects that are instantiated from other classes or from functions outside class definitions. There is an exception to this, which is discussed later in this tutorial. Writing operator functions as class member functions simplifies the content of the operator member functions, since the private data members can be directly accessed.

The assignment operator is the simplest of the three operator functions. The values stored in the x and y components of the vector on the right hand side (rhs) of x=y are assigned to the data members of the object (which is on the left hand side of this equation). The `this` pointer is used to refer to this specific instantiation of the class. The addition and subtraction operators use the assignment operator to create a copy of the object before the values of the data members are changed. When the object is instantiated without arguments, the default constructor parameters are used. Then the assignment function is called to assign the values. For the operators + and - the equations are z=x+y and z=x-y, where all of the object types are the same, y is passed into the member function as the `const` reference `rhs` and x refers to the object for which the member function is called.

The final piece of C++ needed to produce a working example is the `main()` function. Create a new file called `main.cpp` and add the C++ code at the top of the next page.

```
#include "TwoVector.h"
#include <iostream>
#include <cmath>

using namespace std;

int main() {
  TwoVector vec1(3.,4.); // Using the 3,4,5 triangle.
  TwoVector vec2 = vec1; // Copy vec1
  std::cout << "vec2{x=" << vec2.x()
            << ", y=" << vec2.y() << "}" << std::endl;
  std::cout << "vec1.resultant()=" << vec1.resultant() << std::endl;
  std::cout << "vec2.angle()=" << (vec2.angle()/M_PI)*180. << " degrees" << std::endl;
  vec2.rotate(M_PI/2.0); // Rotate anti-clockwise by 90 degrees
  std::cout << "After rotation vec2{x=" << vec2.x() << ", y=" << vec2.y()
            << "}, vec2.angle()=" << (vec2.angle()/M_PI)*180. << " degrees" << std::endl;
  vec1 = vec1 - vec2;

  std::cout << "vec1-vec2 = {x=" << vec1.x()
            << ", y=" << vec1.y() << "}" << std::endl;
  return 0;
}
```

The `main()` function makes use of the three, four, five triangle, to help to make the value of the resultant and the angle more intuitive during debugging. The example creates a vector called `vec1` that has an x component of three and a y component of four. This implies that the resultant is five. The values in `vec1` are then assigned to the values of `vec2`, which is therefore a numerical copy of the first vector. The components of `vec2` are printed, the resultant of `vec1` is printed and the angle in the x-y plane is printed for `vec2`. The function `angle()` returns the value in radians, which is then converted into degrees before being printed on the screen. The vector `vec2` is then rotated by 90 degrees and the resulting angle in the x-y plane is printed. Finally, `vec2` is subtracted from `vec1` and the resulting x and y components are printed on the screen. Once the operator functions have been written, the mathematical usage of the operators becomes intuitive.

To complete the example program and produce an executable that can be run, create a file called `Makefile` in the same directory as the other C++ source files and add:

```
CC=g++
TARGET=op
OBJECTS=main.o TwoVector.o

$(TARGET): $(OBJECTS)
  @echo "** Linking Executable"
  $(CC) $(OBJECTS) -o $(TARGET)

clean:
  @rm -f *.o *~

veryclean: clean
  @rm -f $(TARGET)

%.o: %.cpp
  @echo "** Compiling C++ Source"
  $(CC) -c $(INCFLAGS) $<
```

where the lines should be indented by single tab characters and there should be no spaces in front of any other lines. Save the file and then type `make` to build the executable and `./op` to run it. More information on Makefiles can be found in Issue 7 of The MagPi.

Output stream operators

In the `main()` function in the last example, each component of the vectors is printed by retrieving the value of the component and then printing it to the screen. While this works, implementing these function calls in many places can quickly become a waste of time. Therefore, writing a function that allows an object to be printed directly, e.g.

```
std::cout << vec1 << std::endl;
```

may save time. Unlike the mathematical functions, this operator function is not part of the class. However, since it relates to the class it is intuative to put it into the same header file as the `TwoVector` class definition. Open the `TwoVector.h` header file and modify the file to include the two output stream lines given below:

```
  friend std::ostream& operator<<(std::ostream& os, const TwoVector& vec); // New line to add

private:
  double m_x; // x component of the vector
  double m_y; // y component of the vector
};
std::ostream& operator<<(std::ostream& os, const TwoVector& vec); // New line to add
```

Then add the `iostream` header file at the top:

```
#ifndef TWOVECTOR_H
#define TWOVECTOR_H
#include <iostream>
```

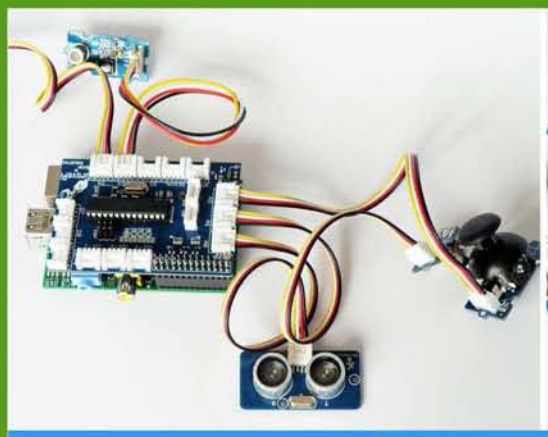
The output stream operator function is not part of the class `TwoVector`. To print the values stored in the `private` data members of the `TwoVector` object, the output stream function needs to access these values. While this is possible by calling the `x()` and `y()` functions to retrieve these values, there is a small overhead for these function calls. Therefore, to make the code slightly simpler, the output stream function is defined as a `friend` within the class declaration. This means that it will be able to access the `private` data members as if they were `public` data members. The `friend` keyword can be used with classes as well as functions. The `friend` keyword also enables access to `protected` functions or data members and `private` functions.

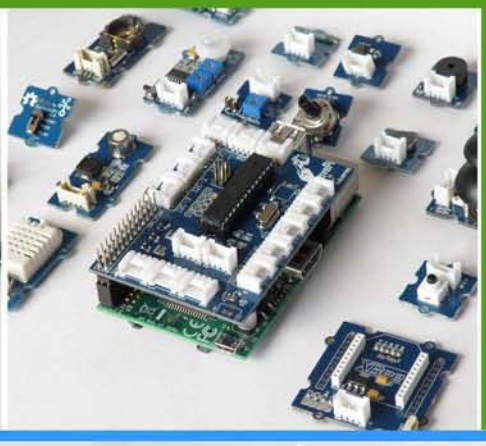
Now open the `TwoVector.cpp` file and at the end of the file add:

```
std::ostream& operator<<(std::ostream& os, const TwoVector& vec){
  os << "{" << vec.m_x << "," << vec.m_y << "}";
  return os;
}
```

This is the implementation that prints the values into the stream. Next, try replacing the lines in the `main.cpp` file that print the components with:

```
std::cout << vec1 << std::endl;
```



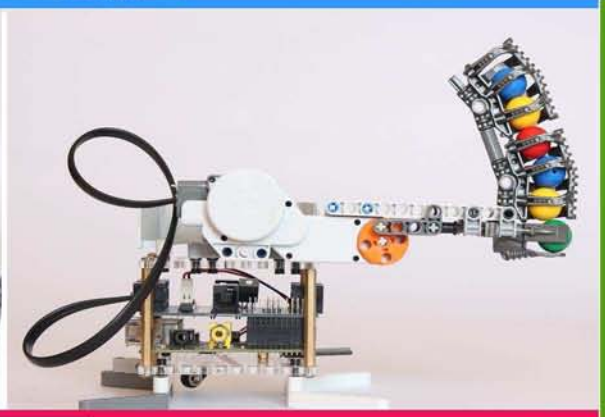


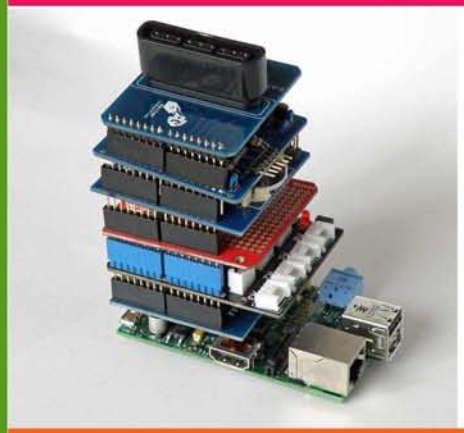
GrovePi

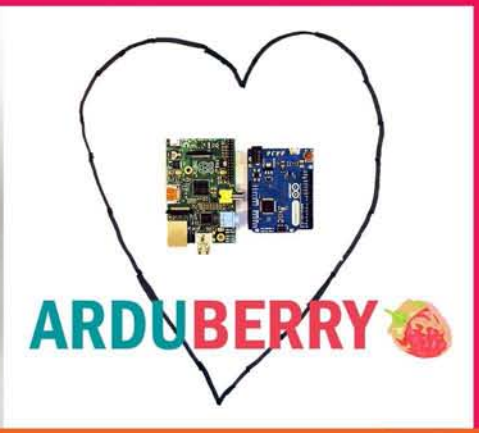
Connect Hundreds of Sensors to your Raspberry Pi

BrickPi

Turn your Raspberry Pi into a LEGO© Robot





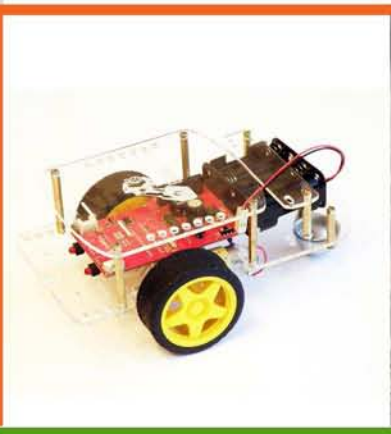


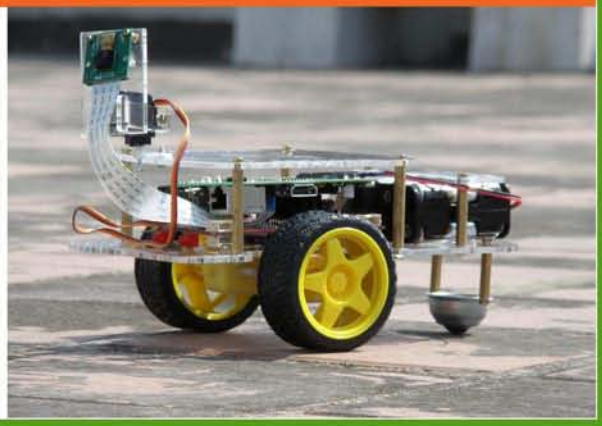
Arduberry

Unite the Raspberry Pi and Arduino

GoPiGo

Turn Your Raspberry Pi into a Robot





dexterindustries.com

MagPi Readers! Use the code "MagPi14" for a 10% discount in our store.

PRINT EDITION AVAILABLE WORLDWIDE

The MagPi is available for FREE from http://www.themagpi.com, from The MagPi iOS and Android apps and also from the Pi Store. However, because so many readers have asked us to produce printed copies of the magazine, we are pleased to announce that printed copies are now regularly available for purchase at the following Raspberry Pi retailers...

Americas

adafruit INDUSTRIES

EMEA

MODMYPI

AUTHENTIC SWAG EDUCATING SINCE '12

The Pi Hut

Pi Supply

AsiaPac

AusPi Technologies Australia

Have Your Say...

The MagPi is produced by the Raspberry Pi community, for the Raspberry Pi community. Each month we aim to educate and entertain you with exciting projects for every skill level. We are always looking for new ideas, opinions and feedback, to help us continue to produce the kind of magazine you want to read.

Please send your feedback to editor@themagpi.com, or post to our Facebook page at http://www.facebook.com/MagPiMagazine, or send a tweet to @TheMagP1. Please send your article ideas to articles@themagpi.com. We look forward to reading your comments.