-THE PiTROL-

control your Pi with the PiTrol

"Could this be your ~~first~~ second step to becoming the **next** generation of Computer Games **Designer?**"

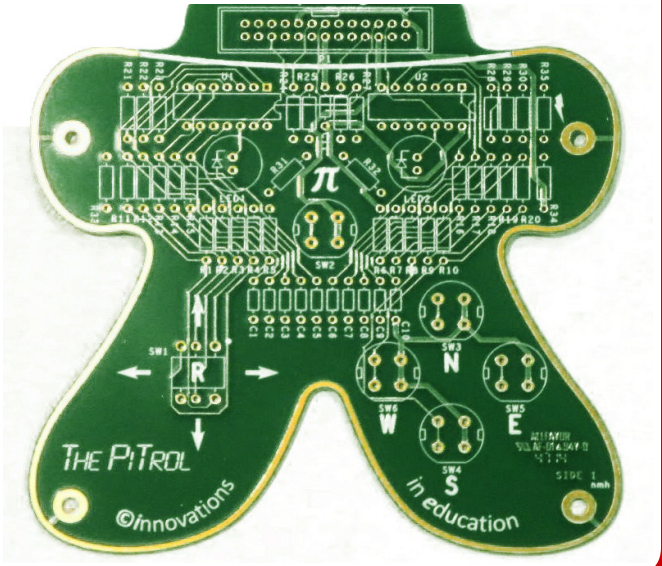
An *innovations in education* Product

Contents

The PiTrol – *ConTrol Your Pi with The PiTrol!*

The PiTrol is a π shaped Games Controller for your **Raspberry Pi®**!
Connect it by Ribbon Cable to the GPIO Connector on your **Raspberry Pi**
then control your games with
a **JoyStick** (Up, Down, Left, Right & Raspberry),
five **PushButtons** (North, South, East, West & Pi)
and 2 **LEDs** (Yellow & Green).
The PiTrol is unlike other games controllers!
The PiTrol is designed to ConTrol the games ***written by you!***

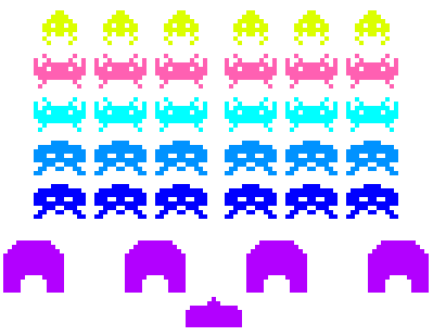
Assemble *The PiTrol* PCB



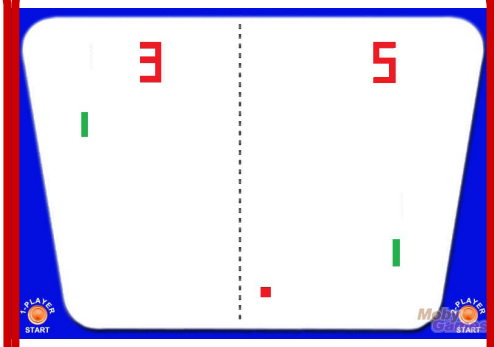
Write the Python game...

After that it is up to you!

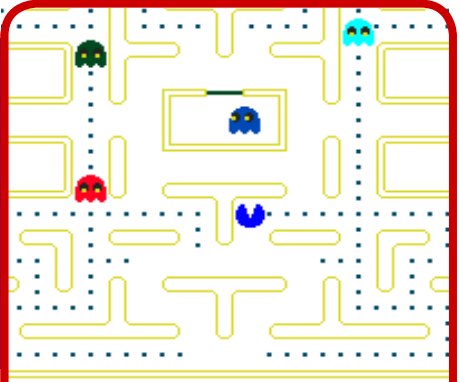
Bring other Python Games to Life with *The PiTrol* !



Convert Python Games to be Controlled by *The PiTrol* !



Enhance Python Games to use more Controls of *The PiTrol* !!



Write new Python Games especially for *The PiTrol* !!!

SOLDERING

All components are fitted on the top surface of ***The PiTrol*** Printed Circuit Board (PCB) and soldered on the underside. To assemble the kit you will need the following:

- ***Soldering Iron***
- ***Solder***
- ***Wire Cutters***
- ***Small Pliers***

TIP! - To ease assembly, fit and solder the smallest components first, followed by the next tallest and so on. The Resistors and Capacitors can be fitted either way round but P1, U1-2, LED1-2, and SW1-6 *MUST BE FITTED AS SHOWN !!!*

TIP! - Solder *just* the 2 opposite corner pins of P1, U1-2, and SW1-6 then turn the board over and *double check* they are the right way round *and* are flush with the surface of the PCB before soldering the other pins - *Once more than 2 pins of these components are soldered they are difficult to remove!!!*

U1 & U2 are CMOS Integrated Circuits (ICs)so can be damaged by electrostatic discharge (ESD)! To protect them from ESD damage they are supplied in conductive foam.

TIP! - Keep the ICs in their conductive foam until you are ready to solder them and avoid nylon clothing or nylon carpet when soldering!

Once assembled and plugged into your Raspberry Pi the ESD ring around the edge of ***The PiTrol***'s PCB will harmlessly dissipate any static charge on your hands when you touch it, thus helping to protect these Integrated Circuits from electrostatic damage. See "***Understanding the Electronics***" for an explanation of how the ESD ring works.

SWITCHES

The PiTrol uses high quality PushButton Switches and a JoyStick of the type used as a navigation switch on DVD Players, etc. On ***The PiTrol*** the switches are exposed with no protective plastic case (unless you fit the optional Perpsex Front Panel)

TIP! - Don't be too rough with *The PiTrol*!

PYTHON

The PiTrol uses the Raspberry Pi**'s GPIO** so you must be running as "Super User". So when you start the Raspbian IDE GUI type `sudo startx`

TIP! - Download the *Seven Segments of Pi* Instructions Manual from www.SevenSegmentsOfPi.com and read the Chapter "Running Python Test Program".

The test program `PiTrol_test.py` is written to run with the Python Editor **IDLE** (known as **Python2** in the lastest version of Raspbian). If you try to run it with **IDLE3/Python3** you will get

```
SyntaxError - invalid syntax
```

To use **IDLE3/Python3** you would need to change they program to use Python3 syntax.

TIP! - Read the "Understanding Python", "Python Troubleshooting" and "Python Debugger" Chapters of the *Seven Segments of Pi* Instructions Manual.

SOUND EFFECTS

There can be problems getting Sound Effects to work on the Raspberry Pi as it can depend on what monitor you are using and even what monitor cable you are using!

TIP! - If your Sound Effects don't work, read the "Troubleshooting Sound Effects" Chapter of the *Seven Segments of Pi* Instructions Manual.

FILE HANDLING

Until you make a Backup the only copy of your software is on your SD Card and SD Cards do occasionally fail!

TIP! - Take regular Backups! Read the "Taking Backups and Transferring Files onto your Raspberry Pi" Chapter of the *Seven Segments of Pi* Instructions Manual.

While developing your software you might sometimes wish you could go back to your last working version!

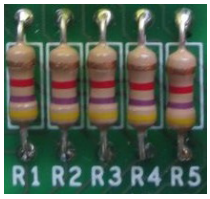
TIP! - Once you have `step1.py` working save the file as `step2.py` before making any more changes. Then you can always go back to `step1.py` if `step2.py` goes all wrong!

Components

Qty	Part Description	PCB Ref	Assembly Notes, etc
1	The PiTrol PCB		Fit components on top side. Solder on under side
20	Resistor 4k7 ohm	R1-20	Marked YELLOW, PURPLE, RED, GOLD
12	Resistor 330 ohm	R21-32	Marked ORANGE, ORANGE, BROWN, GOLD
3	Resistor 680k ohm	R33-35	Marked BLUE, GREY, YELLOW, GOLD
11	Capacitor 100nF	C1-11	Can be fitted either way round
2	Hex Schmitt Inverter IC 74AHC14N	U1-2	! Fit with 'D' in plastic to the right. ESD sensitive!
1	Boxed Header 26 Way	P1	! Fit with cutout towards top of PCB
1	LED Yellow 10mm	LED1	! Fit with short lead (Cathode) towards top of PCB
1	LED Green 10mm	LED2	! Fit with short lead (Cathode) towards top of PCB
1	JoyStick 5 Way	SW1	Snap fit into PCB. It will only fit one way round
1	PushButton Switch Red	SW2	! Fit with flat towards bottom of PCB
1	PushButton Switch Blue	SW3	! Fit with flat towards bottom of PCB
1	PushButton Switch White	SW4	! Fit with flat towards bottom of PCB
1	PushButton Switch Green	SW5	! Fit with flat towards bottom of PCB
1	PushButton Switch Yellow	SW6	! Fit with flat towards bottom of PCB
1	JoyStick KeyCap		Slide onto the JoyStick Spindle
1	JoyStick Raspberry Label		Stick in centre of the top of the JoyStick KeyCap
4	Self Adhesive Feet		Fit on underside of the PCB in positions marked
1	Rainbow Ribbon Cable 26 Way		Connects from The PiTrol P1 to the Raspberry Pi
1	26 Way GPIO Adpater		For connection to the Raspberry Pi 40 way connector

Assembly Instructions

Fit and Solder R1-20



R1-20 are 4.7k ohm Resistors marked with coloured bands YELLOW, PURPLE, RED, GOLD. (See "***Understanding the Electronics***" in the ***Seven Segments of Pi*** Instructions Manual for a description of the resistor colour coding scheme. Resistors can be fitted either way round but fitting them all the same way makes it look neater! Once soldered crop the leads using your Wire Cutters so that no more than about 2mm protrudes on the underside.

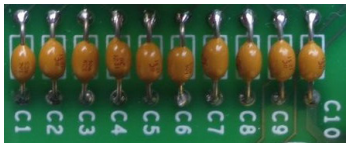
Fit and Solder R21-32

R21-32 are 330 ohm Resistors marked with coloured bands ORANGE, ORANGE, BROWN, GOLD.

Fit and Solder R33-35

R33-35 are 680k ohm Resistors marked with coloured bands BLUE, GREY, YELLOW, GOLD.

Fit and Solder C1-11

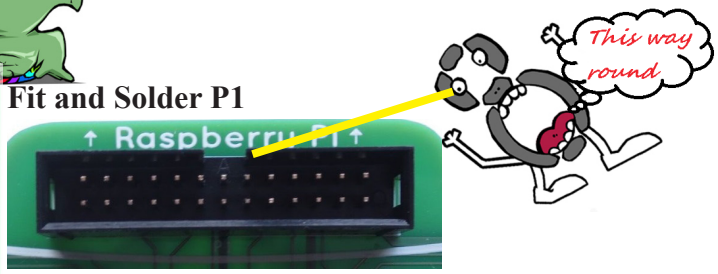


C1-11 are 100nF Capacitors and can be fitted either way round

Fit and Solder U1 & U2

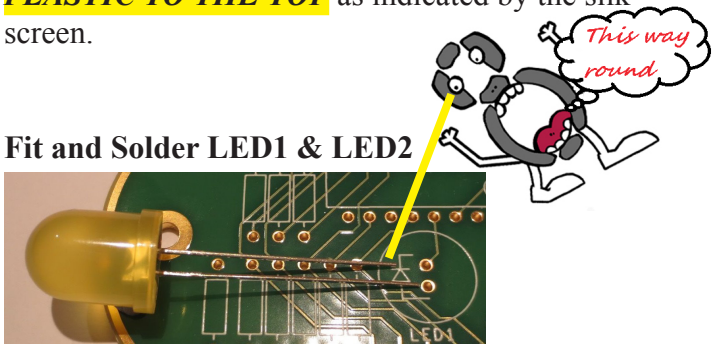
U1 & U2 are Hex Schmitt Inverting Buffer Integrated Circuits (IC's). ***THEY MUST BE FITTED WITH THE 'U' SHAPED CUT-OUT IN THEIR PLASTIC LIDS TO THE RIGHT*** as indicated by the 'U' shape white 'silk screen' marking on the PCB.

Fit and Solder P1



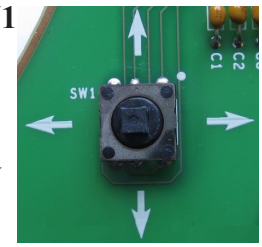
P1 is the Ribbon Cable Connector which ***MUST BE FITTED WITH THE CUT-OUT IN THE PLASTIC TO THE TOP*** as indicated by the silk screen.

Fit and Solder LED1 & LED2



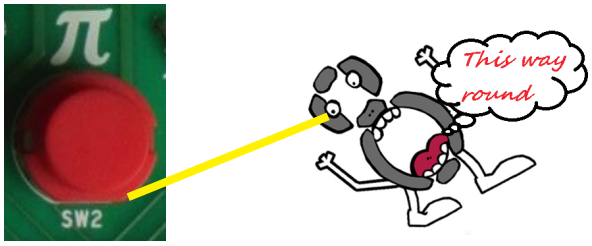
LED1 is a 10mm **Yellow** and LED2 is a 10mm **Green** Light Emitting Diode (LED). They have two leads with one lead shorter than the other. The shorter lead is the Cathode. **FIT WITH THE SHORTER LEAD (the Cathode) TOWARDS THE TOP OF THE PCB** as indicated by the silk screen. Fit the LEDs flush against the PCB

Fit and Solder JoyStick SW1



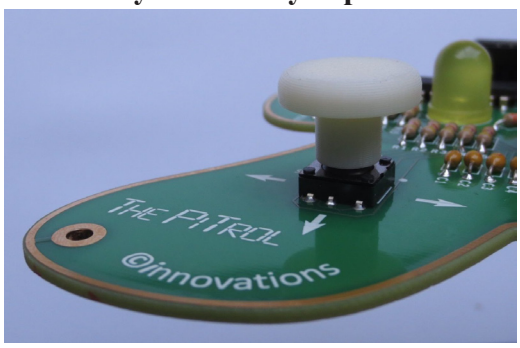
SW1 is the 5 Way JoyStick. It has 6 pins but will only fit into the PCB one way round as shown. Its pins are slightly bent and are designed to snap fit into the PCB.

Fit and Solder PushButton Switches SW2-6



SW2 is the Red “Pi” PushButton Switch.
SW3 is the Blue “North” PushButton Switch.
SW4 is the White “South” PushButton Switch.
SW5 is the Green “East” PushButton Switch.
SW6 is the Yellow “West” PushButton Switch.
THEY MUST BE FITTED WITH THE FLAT PLASTIC SIDE TO THE BOTTOM as indicated by the silk screen.

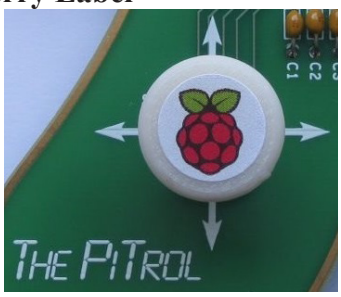
Fit the JoyStick’s KeyCap



The plastic JoyStick’s KeyCap slides onto the JoyStick Spindle. It is tight friction fit so take care not to damage the JoyStick when fitting the KeyCap!

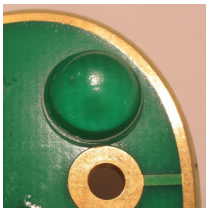
Fit the JoyStick’s Raspberry Label

Fit the self adhesive Raspberry Label centrally on the top surface of the JoyStick KeyCap as shown.



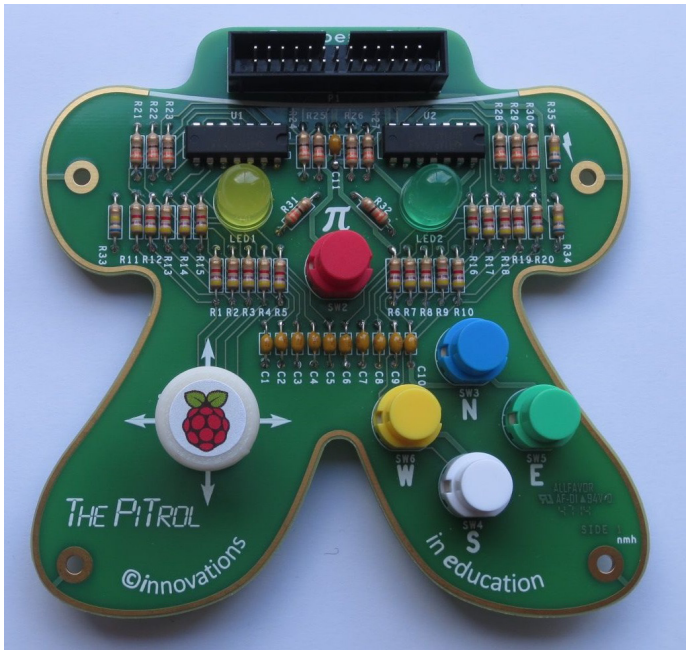
Fit the four Self Adhesive Feet

These fit on the underside of the PCB in the positions marked by four circles on the silk screen.



Fully assembled

The PiTrol PCB when fully assembled looks like this.



Testing and Troubleshooting

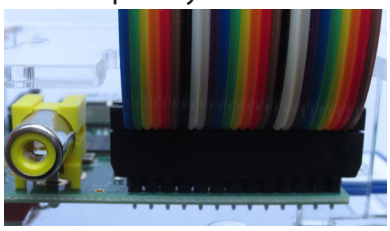
Before attaching ***The PiTrol*** to your Raspberry Pi check the following:-

Checks:-
C1) Are **P1**, **U1-2**, **LED1-2** and **SW2-6** fitted the correct way round?
C2) Are all pins soldered?
C3) Are there any solder shorts between pins?
C4) Are any Resistor or Capacitor wires touching adjacent components?

If everything looks OK you are ready to attach it to your Raspberry Pi.

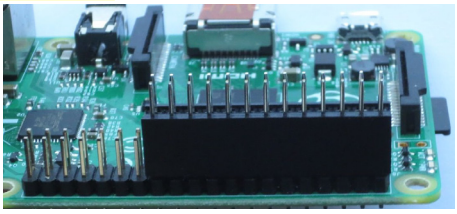
Raspberry Pi Model A or Model B (26 Way GPIO Connector)

Plug one end of the Ribbon Cable into Connector P1 of ***The PiTrol***. Plug the other end into the 26 Way GPIO Header of your Raspberry Pi. ***The PiTrol*** connector is shrouded so the ribbon cable will only fit in it one way round. The Raspberry Pi connector is unshrouded so take care aligning the Ribbon Cable connector with the Raspberry Pi connector pins.

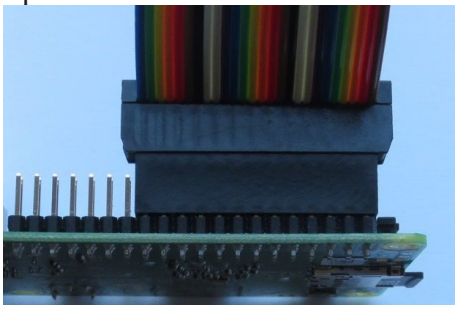


Raspberry Pi Model B+ or Model 2 (40 Way GPIO Connector)

To avoid bending pins on 40 Way GPIO Connector use the 26 Way GPIO Adapter plugged into the ***right hand end 26 pins*** of the 40 Way GPIO Header



Plug one end of the Ribbon Cable into the connector on ***The PiTrol***. Plug the other end into the 26 way GPIO Adapter.



or use ***PiDapter40*** from ***innovations in education*** (see Page 15)

Plug a Raspbian SD Card (or ***Seven Segments of Pi*** "***SSPi***" SD Card) into your Raspberry Pi and power it up.

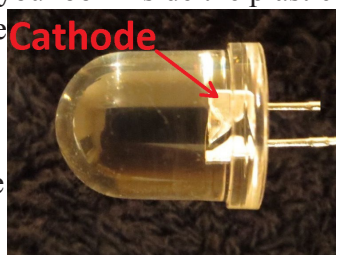
The PiTrol should power up with both LEDs illuminated. If so, you are ready to run the "**Python**" test program as described on the Page 8.

If not, follow the troubleshooting tips below to find the assembly problem:-

Troubleshooting Tips:-
T1) Firstly re-check points C1, C2, C3 & C4 listed above.

T2) If your Raspberry Pi fails to boot it could be that your Power Supply cannot supply the extra current of about 20mA needed by ***The PiTrol***. It is recommended that you use a Power Supply that can supply 5V at a current of 1A (1000mA) or more.

T3) If nothing illuminates there could be a problem with the power connection so check the Ribbon Cable is plugged in correctly at both ends and check the soldering of P1 Connector pins 1, 2, 4 & 6, U1 & U2 pins 7, 12, 13 & 14. Check the soldering of LED1 & 2 and Resistors R31 & R32. The other reason could be that LED1 & 2 have been fitted the wrong way round! Checking that the LEDs are fitted the right way round is tricky once the leads have been cropped! However if you look inside the plastic lens you will see the Anode and Cathode terminals are shaped as shown in the photo. The Cathode needs to be towards the top of the PCB.



T4) If the JoyStick or any of the PushButtons fail to work, check your soldering! Use the Circuit Diagram of ***The PiTrol*** to help work out which components to check. So, for instance, if the Blue "North" PushButton fails to work the problem is likely to be caused by a soldering fault with SW3, R7, R17, C7, U2 pin 3, U2 pin 4, R27 or P1 pin 13.

For any other problems read the section on "**Understanding the Electronics**". This might help you work out which other connection(s) may be causing the fault.

GPIO Used by The PiTrol

The PiTrol controls Python Games via the Raspberry Pi's GPIO. The Table below lists the Switches and LEDs on ***The PiTrol*** and the GPIO Numbers they are connected to. The "**Python Name**" is the name assigned to these GPIO in the Python test program.

Component	PCB Ref	Switch	Python Name	GPIO.BOARD	GPIO.BCM	I/O
JoyStick	SW1	↑	Up	16	23	Input
JoyStick	SW1	↓	Down	18	24	Input
JoyStick	SW1	←	Left	22	25	Input
JoyStick	SW1	→	Right	15	22	Input
JoyStick	SW1	Centre	Raspberry	19	10	Input
PushButton	SW2	Red	Pi	8	14	Input
PushButton	SW3	Blue	North	13	27	Input
PushButton	SW4	White	South	7	4	Input
PushButton	SW5	Green	East	12	18	Input
PushButton	SW6	Yellow	West	11	17	Input
LED	LED1	Left	Yellow	5	3	Output
LED	LED2	Right	Green	3	2	Output

In Python the GPIO can be referenced by the Raspberry Pi GPIO Board Header Physical Pin Number or by the Broadcom (BCM) Number.

To use the GPIO Board Header Physical Pin Numbers you have to use the Python command

```
GPIO.setmode(GPIO.BOARD)
```

Or to use the BCM Numbers you have to use the Python command

```
GPIO.setmode(GPIO.BCM)
```

Either can be used but the Python test program uses:-

```
GPIO.setmode(GPIO.BOARD)
```

Controlling GPIO from Python

The ***Seven Segments of Pi*** Instructions Manual describes how to write Python Software to configure GPIO as Inputs or Outputs and use them to either sense a PushButton, when an Input, or illuminate an LED when an Output. This Manual can be downloaded from www.SevenSegmentsOfPi.com

Just like the ***Seven Segments of Pi***, ***The PiTrol*** has been designed so that when reading an Input in Python, "`True`" means the **PushButton** is pressed and "`False`" means it is *not* pressed. Similarly for Outputs, "`True`" turns the **LEDs** 'on' and "`False`" turns it 'off'. (***Note*** that the **LEDs** will also be 'on' when the GPIO are unconfigured or configured as inputs hence both **LEDs** should be 'on' when the Raspberry Pi is first powered 'on').

Python Test Program for The PiTrol

Download the zip file containing the Python program **`PiTrol_test.py`** from www.ThePiTrol.com. Save it and unzip it in directory:-

```
/home/pi/ThePiTrol
```

Use **`PiTrol_test.py`** to verify that ***The PiTrol*** is fully working. It is also useful as an example of how to configure the GPIO for ***The PiTrol***!

Open the program in **IDLE** then select…

```
Run > Run Module (or just press "F5")
```

If ***The PiTrol*** is working correctly the Yellow and Green LEDs should alternately flash 'on' and 'off'. Then when you press a PushButton Switch or move the JoyStick the "**Python Name**" for that button should be printed on the Console Terminal. Note that the JoyStick is a 5-Way version. Click down on the centre of the JoyStick to activate the "**Raspberry**" button!

If any of the switches fail to work read the "***Troubleshooting Tips***" above to help find the fault.

Opposite is the Python software for **`PiTrol_test.py`** Highlighted in yellow are the lines you will need to add to your Python Games to control them from ***The PiTrol***!


```
#################################################################
# The PiTrol Test Program - PiTrol_test.py                      #
#################################################################
# Description:-                                                 #
# Run this program to test that the The PiTrol is fully working!#
# The program prints Up, Down, Left, Right, Raspberry, Pi,      #
# North, South, East or West on the *Python Shell* Window when a#
# PushButton Switch is pressed or the JoyStick is moved         #
# It also flashes the Yellow and Green LED's alternately        `#
#################################################################
#!/usr/bin/env python    #allows program to be run from command line
import time              #time package allows programmable delays in the software
import RPi.GPIO as GPIO  #RPi.GPIO package allows control of GPIO by software
GPIO.setmode(GPIO.BOARD) #Sets RPi.GPIO package to number GPIO by their Raspberry Pi Connector pin number
GPIO.setwarnings(False)  #Disables GPIO Warning Messages

# Assign the GPIO the name of the JoyStick direction it controls
Up         = 16   #GPIO for Up JoyStick Switch Input
Down       = 18   #GPIO for Right JoyStick Switch Input
Left       = 22   #GPIO for Left JoyStick Switch Input
Right      = 15   #GPIO for Down JoyStick Switch Input
Raspberry = 19    #GPIO for Centre JoyStick Switch Input

# ...and make these GPIO Inputs
GPIO.setup(Up,         GPIO.IN)
GPIO.setup(Down,       GPIO.IN)
GPIO.setup(Left,       GPIO.IN)
GPIO.setup(Right,      GPIO.IN)
GPIO.setup(Raspberry, GPIO.IN)

# Assign the GPIO the name of the PushButton Switch it controls
Pi         =  8   #GPIO for Red Pi PushButton Switch Input
North      = 13   #GPIO for Blue North PushButton Switch Input
South      =  7   #GPIO for White South PushButton Switch Input
East       = 12   #GPIO for Green East PushButton Switch Input
West       = 11   #GPIO for Yellow West PushButton Switch Input

# ...and make these GPIO Inputs
GPIO.setup(Pi,    GPIO.IN)
GPIO.setup(North, GPIO.IN)
GPIO.setup(South, GPIO.IN)
GPIO.setup(East,  GPIO.IN)
GPIO.setup(West,  GPIO.IN)

# Assign the GPIO the name of the LED it controls
YellowLED = 5
GreenLED  = 3

# ...and make these GPIO Outputs
GPIO.setup(YellowLED, GPIO.OUT)   #GPIO for Yellow LED Output
GPIO.setup(GreenLED,  GPIO.OUT)   #GPIO for Green LED Output

poll = .1    # Define Constant for PushButton 'poll' rate
delay = 1    # Define Constant for Count delay
FLASH = True # FLASH is toggled True and False to flash LED's

# Start of The PiTrol Test Program Execution
print "Testing The PiTrol !!!"  # Printed on Raspberry Pi *Python Shell* Window
while True :
    if FLASH == True:                  # If FLASH is True...
        GPIO.output(YellowLED, True)   # Turn Yellow LED 'on'
        GPIO.output(GreenLED, False)   # Turn Green LED 'off'
        FLASH = False                  # then Toggle FLASH to False
    else:                              # else FLASH must be False, so...
        GPIO.output(YellowLED, False) # Turn Yellow LED 'off'
        GPIO.output(GreenLED, True)    # Turn Green LED 'on'
        FLASH = True                   # Toggle FLASH to True

    UP         = GPIO.input(Up)        # Check all Switch GPIO inputs
    DOWN       = GPIO.input(Down)
    LEFT       = GPIO.input(Left)
    RIGHT      = GPIO.input(Right)
    RASPBERRY = GPIO.input(Raspberry)
    PI         = GPIO.input(Pi)
    NORTH      = GPIO.input(North)
    SOUTH      = GPIO.input(South)
    EAST       = GPIO.input(East)
    WEST       = GPIO.input(West)

    if UP         == True:             # if JoyStick is in Up direction
        print "Up"                     # Print "Up" on *Python Shell* Window
    if DOWN       == True:             # if JoyStick is in Down direction
        print "Down"                   # Print "Down" on *Python Shell* Window
    if LEFT       == True:             # etc
        print "Left"
    if RIGHT      == True:
        print "Right"
    if RASPBERRY == True:
        print "Raspberry"
    if PI         == True:
        print "Pi"
    if NORTH      == True:
        print "North"
    if SOUTH      == True:
        print "South"
    if EAST       == True:
        print "East"
    if WEST       == True:
        print "West"
    time.sleep(poll) # wait 0.1 of a second before checking all Switches again
```

Seven Steps to PiTrol Wormy

The Raspbian SD Card (or the ***Seven Segments of Pi*** "SSPi" SD Card) has a directory:-

`/home/pi/python_games`

In this directory you will find the Python game **'`wormy.py`'**

Open and Run the game!

You control "wormy" using the Up, Down, Left & Right keys on your keyboard with the aim of "eating" as many red "apples" as possible without going off the edge of the screen!

It's quite a good game! And credit goes to Al Sweigart who wrote the game and made it freely available on the Raspberry Pi, but now follow the **Seven Steps to PiTrol Wormy** and see how ***The PiTrol*** brings this game to life!

Copy **`wormy.py`** into directory

`/home/pi/ThePiTrol`

And rename it **`PiTrol_wormy_step1.py`** (It's a good idea to give each step a new file name in case you need to go back one step!)

PiTrol_wormy_step1.py

Your first task is to modify the Python software so that 'wormy' is controlled by ***The PiTrol***'s JoyStick instead of the keyboard.

Copy and Paste the yellow highlighted section from **`PiTrol_test.py`** to **`PiTrol_wormy_step1.py`** to assign the GPIO pin numbers to the Switches and LEDs. Add it just after:-

```
import random, pygame, sys
from pygame.locals import *
```

Then, to detect events on the GPIO, add the following immediately afterwards:-

```
# Detect JoyStick events
GPIO.add_event_detect(Up,         GPIO.RISING)
GPIO.add_event_detect(Down,       GPIO.RISING)
GPIO.add_event_detect(Left,       GPIO.RISING)
GPIO.add_event_detect(Right,      GPIO.RISING)
GPIO.add_event_detect(Raspberry, GPIO.RISING)

# Detect PushButton events
GPIO.add_event_detect(Pi,         GPIO.RISING)
GPIO.add_event_detect(North,      GPIO.RISING)
GPIO.add_event_detect(South,      GPIO.RISING)
GPIO.add_event_detect(East,       GPIO.RISING)
GPIO.add_event_detect(West,       GPIO.RISING)
```

At the beginning of the main game loop "comment out" the section that looks for keyboard events and add the new section highlighted below in yellow that looks for GPIO events.

TIP-A quick way to "comment out" a section is to highlight the section then select

`Format>Comment Out Region`

This adds **`##`** to the start of each line making the lines comments, so they are ignored by the program.

```
    while True: # main game loop
##        for event in pygame.event.get(): # event handling loop
##            if event.type == QUIT:
##                terminate()
##            elif event.type == KEYDOWN:
##                if (event.key == K_LEFT or event.key == K_a) and direction != RIGHT:
##                    direction = LEFT
##                elif (event.key == K_RIGHT or event.key == K_d) and direction != LEFT:
##                    direction = RIGHT
##                elif (event.key == K_UP or event.key == K_w) and direction != DOWN:
##                    direction = UP
##                elif (event.key == K_DOWN or event.key == K_s) and direction != UP:
##                    direction = DOWN
##                elif event.key == K_ESCAPE:
##                    terminate()
##            elif event.type == KEYDOWN:
##                if event.key == K_ESCAPE:
##                    terminate()
# Joystick GPIO
        if GPIO.event_detected(Left) and direction != RIGHT:
                direction = LEFT
        if GPIO.event_detected(Right) and direction != LEFT:
                direction = RIGHT
        if GPIO.event_detected(Up) and direction != DOWN:
                direction = UP
        if GPIO.event_detected(Down) and direction != UP:
                direction = DOWN
```

Run the program!

You still need to press a key on the keyboard to start the game but you should find that ***The PiTrol***'s JoyStick now controls the wormy!

PiTrol_wormy_step2.py

Next modify the Python software so that ***The PiTrol***'s "**Pi**" PushButton starts the game. Also modify the position, colour and wording of the start of game text. Once again the changes are highlighted in yellow below.

In the function

```
def drawPressKeyMsg():
```

change the wording and colour of the start message as follows:-

```
    pressKeySurf = BASICFONT.render('Press Red "Pi" PushButton to Play', True, RED)
```

and change the position of the message to be:-

```
    pressKeyRect.topleft = (WINDOWWIDTH - 350, WINDOWHEIGHT - 30)
```

so it still fits on the screen.

Now change the function that checks for a key press to "comment out" the keyboard check and replace it by checking if the "**Pi**" PushButton has been pressed

```
def checkForKeyPress():
##    for event in pygame.event.get():
##        if event.type == QUIT:       #event is quit
##            terminate()
##        elif event.type == KEYDOWN:
##            if event.key == K_ESCAPE:   #event is escape key
##                terminate()
##            else:
##                return event.key   #key found return with it
    if GPIO.event_detected(Pi):  # Has Pi Pushbutton been pressed?
        return True   # key found return True
    # no quit or key events in queue so return None
    return None
```

PiTrol_wormy_step3.py

Now Flash ***The PiTrol's*** LEDs as the game is playing!

In function

```
def runGame():
```

initialise a variable called 'FLASH' by adding:-

```
    FLASH = True    # used for Flashing LED's during game
```

Then in…

```
def runGame():
    while True: # main game loop
```

add:-

```
        if FLASH == True:
            GPIO.output(YellowLED, True)    # Yellow LED on
            GPIO.output(GreenLED,  False)   # Green LED off
            FLASH = False
        else:
            GPIO.output(YellowLED, False)   # Yellow LED off
            GPIO.output(GreenLED,  True)    # Green LED on
            FLASH = True
```

PiTrol_wormy_step4.py

Make the game full screen. (These settings may need some further modification depending on the monitor you are using). And rather than a Green wormy starting in a random location, make it a Yellow wormy and start it in the middle of the left side.

To change the screen size modify the following settings which set the number of pixels for the window's width and height and the number of pixels for each cell:-

```
WINDOWWIDTH = 1216
WINDOWHEIGHT = 912
CELLSIZE = 38
```

Colours are defined by the brightness of the 3 primary colours Red, Green & Blue thus any colour can be created. Each colour can have any value from 0 to 255. For example try defining a new colour with Maximum Red (255), Maximum Green (255) and Minimum Blue (0) and give it the name **YELLOW** as shown below. You could of course create any other colour!

```
YELLOW     = (255, 255,   0)
```

Then in Function

```
def drawWorm(wormCoords):
```

where it draws the wormy make it **YELLOW**

```
        pygame.draw.rect(DISPLAYSURF, YELLOW, wormInnerSegmentRect)   # wormy is now Yellow
```

And to make the wormy always start in the middle of the left side, in Function

```
def runGame():
```

Comment out

```
##    startx = random.randint(5, CELLWIDTH - 6)
##    starty = random.randint(5, CELLHEIGHT - 6)
```

And set a fixed starting point at an 'x' co-ordinate of '1' and a 'y' co-ordinate equal to half the number of cells. Note **CELLHEIGHT=WINDOWHEIGHT/CELLSIZE**

```
    startx = 1
    starty = (CELLHEIGHT/2)
```

PiTrol_wormy_step5.py

Add Sound Effects! Use these Sound Effects from the ***Seven Segments of Pi*** zip file (which can be can be downloaded from www.SevenSegmentsOfPi.com) or load your own Sound Effects. Refer to the ***Seven Segments of Pi*** Instructions Manual to learn more about adding and troubleshooting Sound Effects.

Define the Sound Effects to be used as follows:-

```
global PHASER, TICK, TWANG, DOH              # Defines the names of some sounds
pygame.mixer.pre_init(44100,-16,2,512)       # Sets Sound parameters (freq,size,channels,buffer)
pygame.init()                                # Initialise pygame package
                                             # Load the sound files:-
PHASER = pygame.mixer.Sound('phaser.wav')    # sound used at start of game
TICK   = pygame.mixer.Sound('tick.wav')      # sound used as wormy worms along
TWANG  = pygame.mixer.Sound('twang.wav')     # sound used when wormy eats an Apple
DOH    = pygame.mixer.Sound('doh.wav')       # sound when wormy falls off the edge of the world!
```

Then to play for example the 'Phaser' Sound Effects add these lines at the appropriate point of the program:-

```
        sound = PHASER    # ...play the sound
        sound.play()
```

I'll leave you to work out where in the program to add the Sound effects!

PiTrol_wormy_step6.py

Start the game more slowly, then speed up each time wormy eats an apple. Also you might want to disable the software that ends the game when wormy 'hits' itself. When using a JoyStick it is too easy to do so!

FPS (Frames Per Second) sets the speed of the game. Try reducing the starting speed from 20 down to 6

```
FPS = 6       # Starting Speed in Frames Per Second
```

Since the speed will be changing during the game it will also be necessary to reset FPS to 6 at the start of each game:_

```
def runGame():
    FPS = 6      # Reset speed to 6 Frames Per Second at start of each new game
```

Then when the wormy eats an apple increase FPS by 1

```
        # check if worm has eaten an apply
        if wormCoords[HEAD]['x'] == apple['x'] and wormCoords[HEAD]['y'] == apple['y']:
            # don't remove worm's tail segment
            FPS = FPS + 1       # Speed up by one FPS each time wormy eats an Apple
```

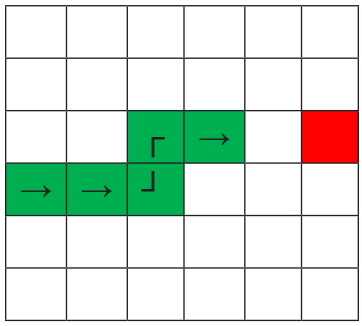
If you also want to disable the software that ends the game when wormy 'hits' itself, "comment out" these lines:-

```
##        for wormBody in wormCoords[1:]:
##            if wormBody['x'] == wormCoords[HEAD]['x'] and wormBody['y'] == wormCoords[HEAD]['y']:
##                return # game over
```

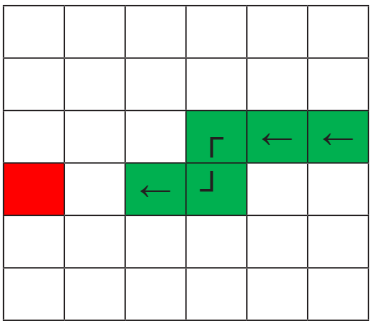
PiTrol_wormy_step7.py

The final step to complete ***PiTrol Wormy*** brings the **North**, **South**, **East** & **West** PushButtons into play! The JoyStick changes wormy's direction, but sometimes wormy is heading in the right direction but is just one row away from the Apple so how about adding a 'nudge' function, controlled by the PushButtons, to 'nudge' wormy by one row!

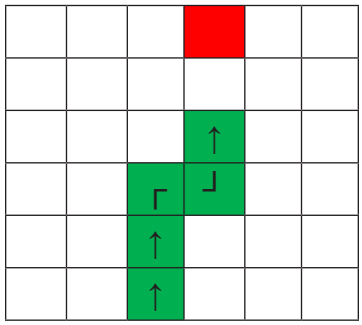
Use the '**North**' PushButtons to 'nudge' wormy up one row:-



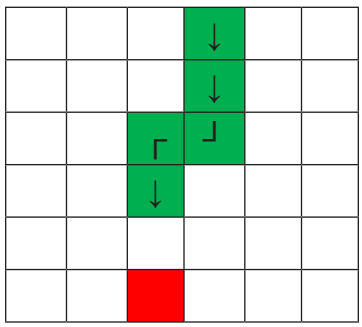
Use the '**South**' PushButtons to 'nudge' wormy down one row:-



Use the '**East**' PushButtons to 'nudge' wormy one row to the right:-



And use the '**West**' PushButtons to 'nudge' wormy one row to the left:-



So this is what the 'nudge' function needs to do. All you have to do now is to write the software!

PiTrol Wormy Extras

You may have your own ideas how you could improve ***PiTrol Wormy*** further, but here are a few suggestions:-

X1) Display 'Todays High Score'

X2) Make Random Speed changes when wormy eats the apple

X3) Press the Raspberry Button to eat the apple

X4) Make the controls work back-to-front so JoyStick '**Up**' makes wormy go down, '**Left**' makes it go right, etc

X5) When wormy goes off the right edge press the '**Pi**' PushButton to go into 'Hyperspace' and reappear on the left edge, or if going off the top reappear at the bottom, etc.

X6) Add a few 'Rotten' Apples which wormy must avoid!

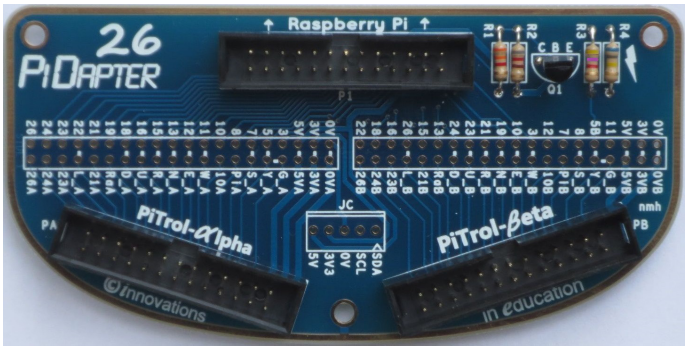
X7) Make the 'Apple' a 'Raspberry' and making it look like the Raspberry Pi Logo!

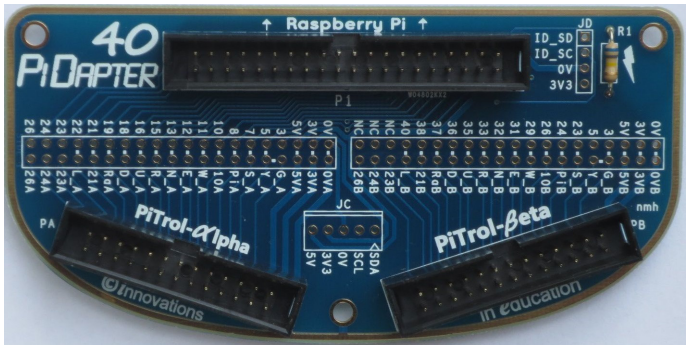
PiDapter for 2 Player PiTrol Games

PiDapter from ***i*nnovations in *e*ducation** allows 2 ***PiTrols*** to connect to the Raspberry Pi so you can write 2 Player Python Games. ***PiDapter*** is available in 2 versions:-

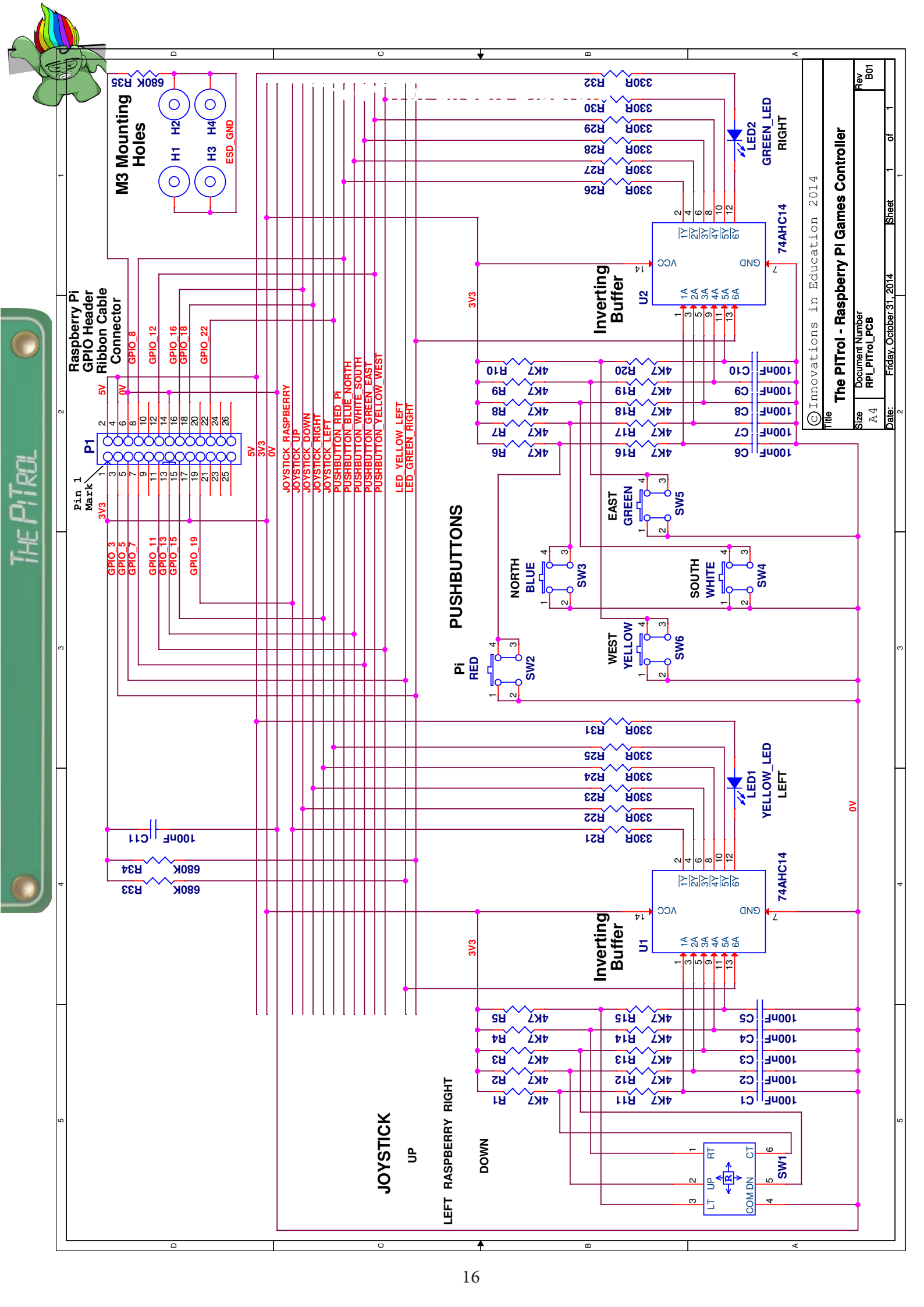
PiDapter26
for **Raspberry Pi Models A or B**
with 26 Way GPIO Connector

PiDapter40
for **Raspberry Pi Models B+ or 2**
with 40 Way GPIO Connector





See www.PiDapter.com


THE PiTROL
M3 Mounting Holes
H1
H2
H3
H4
ESD_GND
680K R35
Raspberry Pi GPIO Header Ribbon Cable Connector
P1
Pin 1 Mark
5V
0V
3V3
GPIO_8
GPIO_12
GPIO_16
GPIO_18
GPIO_22
GPIO_3
GPIO_5
GPIO_7
GPIO_11
GPIO_13
GPIO_15
GPIO_19
JOYSTICK_RASPBERRY
JOYSTICK_UP
JOYSTICK_DOWN
JOYSTICK_RIGHT
JOYSTICK_LEFT
PUSHBUTTON_RED_Pi
PUSHBUTTON_BLUE_NORTH
PUSHBUTTON_WHITE_SOUTH
PUSHBUTTON_GREEN_EAST
PUSHBUTTON_YELLOW_WEST
LED_YELLOW_LEFT
LED_GREEN_RIGHT
PUSHBUTTONS
Pi RED SW2
NORTH BLUE SW3
EAST GREEN SW5
SOUTH WHITE SW4
WEST YELLOW SW6
JOYSTICK
UP
LEFT
RASPBERRY
RIGHT
DOWN
SW1
RT
CT
UP
LT
COM
DN
R
Inverting Buffer
U1
U2
74AHC14
VCC
GND
LED1 YELLOW_LED LEFT
LED2 GREEN_LED RIGHT
330R R21 R22 R23 R24 R25 R31
330R R26 R27 R28 R29 R30 R32
4K7 R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12 R13 R14 R15 R16 R17 R18 R19 R20
100nF C1 C2 C3 C4 C5 C6 C7 C8 C9 C10 C11
680K R33 R34
3V3
0V
©Innovations in Education 2014
Title
The PiTrol - Raspberry Pi Games Controller
Size A4
Document Number RPI_PiTrol_PCB
Rev B01
Date: Friday, October 31, 2014
Sheet 1 of 1

Understanding the Electronics

The PiTrol connects to the Raspberry Pi via a Ribbon Cable plugged onto the Raspberry Pi's GPIO Connector.

The electronics consists of these main components:-

- ***A 5-Way JoyStick***
- ***Five coloured PushButton Switchs***
- ***Two LEDs***
- ***Two Schmitt Inverting Buffer Integrated Circuits (IC's)***

The Raspberry Pi controls ***The PiTrol*** via 12 GPIO (General Purpose Input/Output) signals on the GPIO Connector. GPIO are signals that can be configures as Inputs or Outputs under software control. ***The PiTrol*** uses ten GPIO as an inputs so the software can check if the JoyStick or PushButton Switches have been pressed and two GPIO as outputs so the software can illuminate the two LEDs.

The PiTrol Circuit Description

You don't need to understand the electronic circuit to assemble your kit or write your software however the better you understand the electronics the easier you will find it to fix any assembly errors. It may also help if you are thinking of designing your own electronics to connect to the Raspberry Pi!

Opposite is ***The PiTrol*** circuit diagram.

`SW1` - The JoyStick. The JoyStick has 6 pins. When it is moved right, pin 1 (RT) connects to pin 4 (COM). Pin 4 is the 'Common' pin and is connected to 0 Volts, thus RT will be pulled down to 0 Volts (i.e. Logic '0'). When the JoyStick is released RT is pulled back to 3.3 Volts (i.e. Logic '1') by **`R5`**. Similarly when the JoyStick is moved up, pin 2 (UP) connects to pin 4 (COM) thus UP will be pulled down to 0 Volts. When the JoyStick is moved left, pin 3 (LT) will be pulled down to 0 Volts. When the JoyStick is moved down, pin 5 (DN) will be pulled down to 0 Volts. And when the JoyStick Centre Button is pressed, pin 6 (CT) will be pulled down to 0 Volts.

`SW2-6` - Coloured PushButton Switches. These switches have 4 pins. Pins 1 & 2 are connected within the switch as are pins 3 & 4. When the Red Switch is pressed pins 3 & 4 are connected to pins 1 & 2 thus pulling the voltage on pins 3 & 4 down to 0 Volts (i.e. Logic '0'). When the Red Switch is released **`R6`** pulls the voltage back to 3.3 Volts (i.e. Logic '1'). The other coloured switches are connected in a similar way.

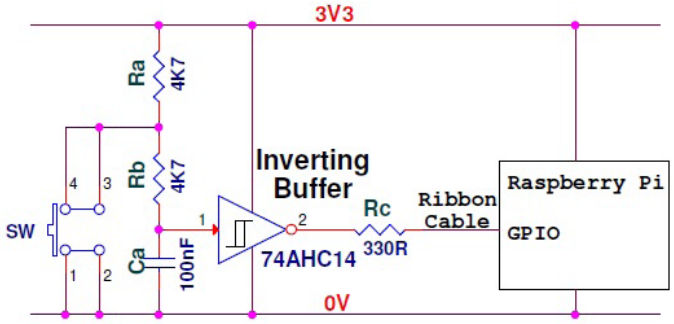
`LED1-2` - 10mm LEDs. The 'Anode' of each LED is connected to 5 Volts via a 330 ohm Resistor so when the 'Cathode' of the LED is driven to 0 Volts by the Inverting Buffer IC about 10mA will flow through the LED and it will illuminate.

`U1-2` - Schmitt Inverting Buffer IC's. These are 74AHC14 Hex Inverting Schmitt Buffers in 14 pin Dual In-Line (DIL) packages. They 'buffer' the switch signals to the Raspberry Pi's GPIO and invert them so that when the switches are pressed the GPIOs are set to Logic '1' and when not pressed they are set to a Logic '0'. Hence in your **Python** '**`True`**' means the switch has been pressed and '**`False`**' mean the switch is not pressed, which is easier to remember! These buffers have '***Schmitt Trigger***' inputs which means that as the voltage on their inputs fall from 3.3 Volts to 0 Volts the inputs will only be 'seen' as a Logic '0' when the voltage falls below about 1.5 Volts but will then not be seen as a Logic '1' until the voltage rises back above about 2.0 Volts. The Buffer's Schmitt Trigger inputs form part of the Switch Debounce circuit and their outputs provide the 10mA drive current needed to illuminate the LEDs.

Switch Bounce

Switches are electro-mechanical components. When a switch is pressed the switch actuator makes electrical contact with the switch terminals. In an ideal switch once contact has been made the actuator would remain in contact with the switch terminal until the switch is released. In a real switch however, when the actuator makes contact with the switch terminals the actuator may 'bounce' several times over a period of many micro seconds, alternately making contact for a few micro seconds then loosing contact for a few micro seconds before eventually settling down in contact. Since the software can 'read' the GPIO input very rapidly it may appear to the software that the switch has been pressed multiple times rather than just once. To alleviate this problem an electronic circuit called a '***Switch Debounce***' circuit is used to help ensure that once the switch is pressed and the GPIO is driven to a Logic '1' it remains at a Logic '1' until the switch is released.

Switch Debounce Circuit. The circuit used by all switches on ***The PiTrol*** is shown below.

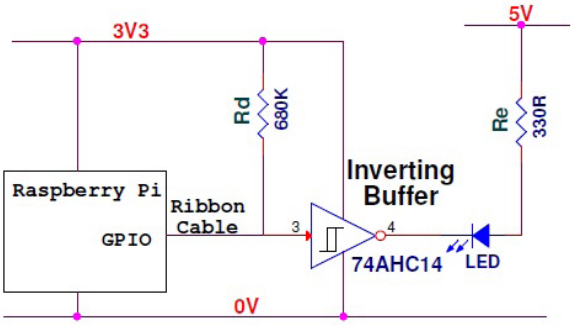


When the switch is ***unpressed*** Resistors 'Ra' and 'Rb' pull the input pin of the Schmitt Inverting Buffer to 3.3 Volts so Capacitor 'Ca' will be charged to 3.3 Volts and the output of the Inverting Buffer will be Logic '0'. When the switch is ***pressed*** Capacitor 'Ca' will discharge via Resistor 'Rb' and Switch 'Sw'. The approximate time taken for it to discharge can be calculated by multiplying the capacitance by the resistance. i.e. 100nF x 4k7 = 470 micro seconds. When the voltage falls below about 1.5 Volts the input to the Inverting Buffer will 'see' this as a Logic '0' so will change its output to Logic '1'. If at this time the switch contacts 'bounce' Capacitor 'Ca' will start to charge back towards 3.3 Volts via 'Ra' and 'Rb'. Since the Inverting Buffer has Schmitt Trigger inputs, the output of the Buffer will not change unless the voltage on its input rises above 2.0 Volts. This will take approximately Ca x (Ra + Rb) x (2.0 Volts - 1.5 Volts)/(3.3 Volts - 1.5 Volts) = 260 micro seconds. Hence, so long as each 'bounce' lasts for no longer than 260 micro seconds, the output of the Inverting Buffer will not change.

Some Switch Debounce circuits don't include 'Rb'. They connect the Switch directly to Capacitor 'Ca' and the input of the buffer. Such a debounce circuit will work, however when the Switch is pressed Capacitor Ca will discharge rapidly with a current of several Amps flowing through the switch for a few nano seconds. Since ***The PiTrol*** connects to the Raspberry Pi via a long Ribbon Cable which acts as an inductor the voltage on '0V' (0 Volts) on ***The PiTrol*** PCB will rise briefly to maybe 2 or more volts before falling back to 0 Volts. This is called 'Ground Bounce' and it may cause the buffer for one of the Switches which has ***not*** been pressed to briefly change its output to a Logic '1' making it look to the software like a second Switch has been pressed! Resistor 'Rb' ensures that Capacitor 'Ca' discharges more slowly hence avoiding this problem of 'Ground Bounce'.

Resistor 'Rc', 330 Ohm in series with the output of the Inverting Buffer, is not part of the debounce circuit. 'Rc' is included to protect the Raspberry Pi's GPIO from damage should you accidentally program the GPIO as an output. If so the Raspberry Pi's GPIO may be outputting a Logic '1' while the Inverting Buffer is outputing a Logic '0'. This will mean current will be drawn from the GPIO. 'Rc' limits this current to approximately 3.3 Volts divided by 330 Ohms = 10 mA, well below the 16 mA maximum current that can safely be drawn from the Raspberry Pi's GPIO thus no damage will occur.

LED Driver Circuit. The circuit used to drive the LEDs on ***The PiTrol*** is shown below.



When the Raspberry Pi's GPIO is configured by your Python software as an output and set to '**`False`**' the GPIO will output a Logic '0', hence the Inverting Buffer will output a Logic '1' i.e. a voltage of approximately 3.3 Volts. Since the LED needs a voltage of at least 2 Volts across it to illuminate, so with only 1.7 Volts across it the LED will be ***unilluminated***.

When the Raspberry Pi's GPIO is configured by your Python software as an output and set to '**`True`**' the GPIO will output a Logic '1', hence the Inverting Buffer will output a Logic '0' i.e. a voltage of approximately 0 Volts. With a voltage of about 2 Volts across the LED this will mean there is approximately 3 Volts across Resistor 'Re', hence a current of approximately 3 Volts divided by 330 Ohms = 9mA will flow through Resistor 'Re' and the LED will be ***illuminated***.

When the Raspberry Pi's GPIO is unconfigured (or is configured as an input) Resistor 'Rd' pulls the input of the Inverting Buffer up to Logic '1' hence when the Raspberry Pi is first powered on both LEDs will be ***illuminated***.

The LEDs are connected to the 5 Volt rail rather than the 3.3 Volt rail to minimise the current drawn by ***The PiTrol*** from the 3.3 Volt rail.

ESD Ring. Integrated Circuits (IC's), particularly CMOS IC's as used on ***The PiTrol***, are susceptable to damage from Electrostatic Discharge (ESD). Modern CMOS IC's have ESD diodes built into their silicon to give some protection but since the electronics on ***The PiTrol*** are exposed the PCB has been designed with an "ESD Ring". This ESD Ring further minimise the risk of damage to the electronics by harmlessly disipating the static electric charge on the fingers of anyone picking up ***The PiTrol***.

The human body can hold a static charge of up to 8k Volts or more. (A static voltage of 4k Volts is enough to generate a spark when you touch something metalic and grounded but even 2k Volts could damage an IC). The human body has a capacitance of about 100pF and resistance of about 1.5k Ohms so a even a 2k Volt static discharge would generate a current of over an Amp for about 100ns!

The ESD Ring is an exposed gold plated copper track running around the whole edge of ***The PiTrol*** PCB on both sides which means it is the first thing you touch when you it pick up!

The ESD Ring is connected to 0 Volts via a 680k Ohm Resistor (R35) so any static charge on your fingers when you first touch ***The PiTrol*** will discharge through R35 but, because the resistance of R35 is so large, it will harmlessly dischage a 2kV static charge with a current of only about 3mA for about 50us. If you then touch the pins of one of the IC's your fingers will no longer hold any static charge so there will be no chance of damaging the IC's.

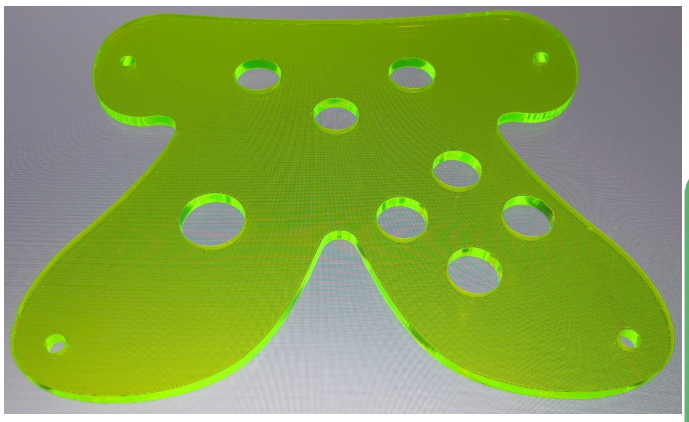
Instruction Manual Updates

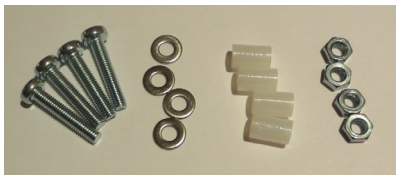
Whilst every effort has been made to ensure these instructions are comprehensive and accurate it is possible there may be errors or omissions. Please let me know of any such errors or omissions via the web site www.ThePiTrol.com.
Your help in doing so is very much appreciated!

I will publish on the web site any corrections or updates as I become aware of them.

Perspex Front Panel

The PiTrol Perspex Front Panel is available as an optional extra along with the screws, nuts, spacers and washers to fix it to ***The PiTrol***. To fix it remove the JoyStick KeyCap (which is friction fit but can be quite tight!) and secure the front panel with the screws nuts, spacers and washers as shown.





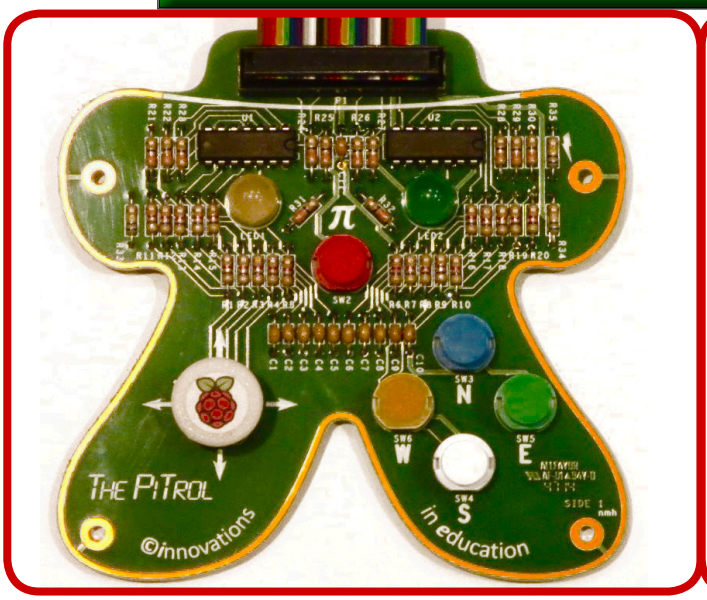
Acknowledgements

I would like to express my gratitude to everyone who has contributed to ***The PiTrol***.
In particular I would like to thank:-

Ryan Beal, Ashley Brown & Graham Mott of Kennet School, Thatcham, UK, who inspired the idea of a games controller for the Raspberry Pi.
Molly & Tim Gudz who inspired the name '***PiTrol***' and tested the early prototypes.
Matt Hunt for designing the graphics and the JoyStick KeyCap.
The PiTrol Beta Testers who have tried out ***The PiTrol*** and sent me comment, corrections and suggestions for improvements!
Al Sweigart (http://inventwithpython.com) for the original **`wormy.py`** game released under a "Simplified BSD" license.

- Nevil Hunt innovations in education -

The Complete PiTrol !



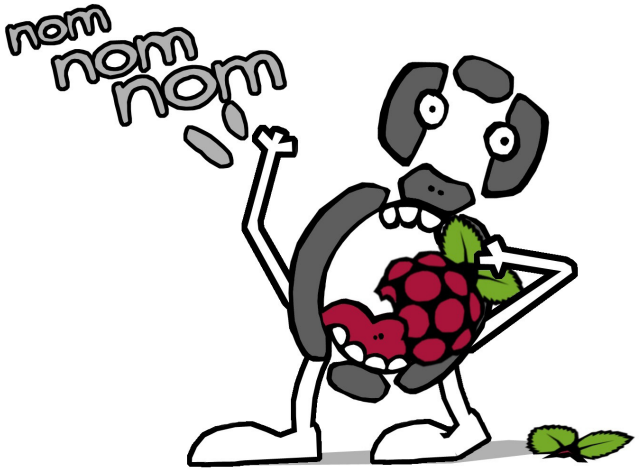
Have Fun while you Learn with The PiTrol...

Second

...and this could be your ~~first~~ step to becoming one of the next generation of Computer Games Designers!

Other Products from *i*nnovations in *e*ducation

Seven Segments of Pi



"Could this be your first step to becoming the **next** generation of Computer Games **Designer?**"

www.SevenSegmentsOfPi.com

PiDapter

"Could this be your ~~first~~ third step to becoming the **next** generation of Computer Games **Designer?**"

www.PiDapter.com

*i*nnovations in *e*ducation

www.ThePiTrol.com