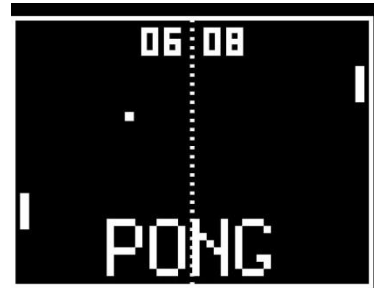


PONG – Use IO (after “Add two extra balls”)

N.B. will not work on a PC! Only on the Raspberry Pi.



The plan is to allow extra balls to appear when a ball goes out of play on an adjacent Pi monitor, using the general purpose I/O (GPIO) on the raspberry Pi to communicate. The following instructions add the code to handle the inputs and outputs.

The “wiringpi” library has all we need in order to use the I/O. Add these lines at the top of the program to include the library and initialize the I/O:

```
import random # Can generate random positions for the pong ball
import wiringpi #to manipulate I/O
import os # To execute the gpio commands from within the program

from pygame.locals import *
from pygame import *

Player1Score = 0
Player2Score = 0

PIN_INPUT_1 = 18
PIN_INPUT_2 = 23
PIN_OUTPUT_1 = 17
PIN_OUTPUT_2 = 22

# Our main game class
class PiPong:

    def __init__(self):
        # initialize wiringpi
        os.system("gpio export 17 out")
        os.system("gpio export 22 out")
        os.system("gpio export 18 in")
        os.system("gpio export 23 in")
        os.system("gpio mode 1 in")
        os.system("gpio mode 1 up")
        os.system("gpio mode 4 in")
        os.system("gpio mode 4 up")
        wiringpi.wiringPiSetupSys();
        # Make the display size a member of the class
```

P.T.O...

Note that the “export” commands use the same numbering system (from the chip manufacturer) as is used in our code. Whereas the “mode” commands use an alternate system (from the creator of the “wiringPi” library) where pin number 1 is equivalent to GPIO 18 and pin number 4 is equivalent to GPIO 23. See <https://projects.drogon.net/raspberry-pi/wiringpi/pins/> for more information and a diagram.

Now add these lines, which make sure that the outputs are off, at the end of the main class `__Init__` function:

```
# record previous player scores so that we know when a ball has gone out
self.PrevPlayer1Score = 0
self.PrevPlayer2Score = 0
# counter for input detection
self.IPCount = 0
# counters to hold output high
self.OP1Count = 0
self.OP2Count = 0
# set outputs off
io.digitalWrite(PIN_OUTPUT_1, 0)
io.digitalWrite(PIN_OUTPUT_2, 0)

def run(self):
```

Now add the following two function calls to the main class run function:

```
# Handle Events
self.handleEvents()

# Handle Inputs
self.handleInputs()

# Handle Outputs
self.handleOutputs()
```

P.T.O...

Now add the `handleInputs` function. It reads the inputs every 5 ticks of the main class. This will be frequent enough to make sure that the input is not missed because the outputs are held for 15 ticks. Note that the inputs have been configured to “pull up” which means that they will be on unless connected to ground:

```
def handleInputs(self):
    if self.IPCount >= 5:
        input1 = io.digitalRead(PIN_INPUT_1) # show ball 2 if this input is off
        if input1 == False:
            self.ball2.show()
        input2 = io.digitalRead(PIN_INPUT_2) # show ball 3 if this input is off
        if input2 == False:
            self.ball3.show()
        self.IPCount = 0
    else:
        self.IPCount = self.IPCount + 1
```

Now add the `handleOutputs` function.

```
def handleOutputs(self):
    if self.OP1Count > 0:
        self.OP1Count = self.OP1Count - 1
        if self.OP1Count == 0:
            io.digitalWrite(PIN_OUTPUT_1, 0)
    elif Player1Score <> self.PrevPlayer1Score:
        self.OP1Count = 15
        io.digitalWrite(PIN_OUTPUT_1, 1)
        self.PrevPlayer1Score = Player1Score

    if self.OP2Count > 0:
        self.OP2Count = self.OP2Count - 1
        if self.OP2Count == 0:
            io.digitalWrite(PIN_OUTPUT_2, 0)
    elif Player2Score <> self.PrevPlayer2Score:
        self.OP2Count = 15
        io.digitalWrite(PIN_OUTPUT_2, 1)
        self.PrevPlayer2Score = Player2Score
```

P.T.O...

All that is needed now is for the wiring and electronics between the Pi computers to be set up.