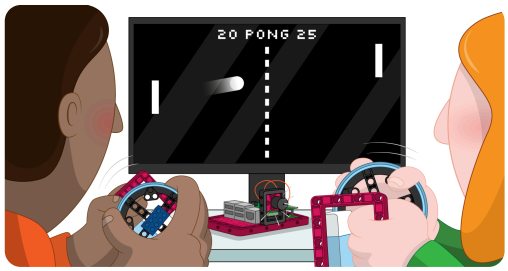
Projects

LEGO® game controller

Create a game controller using LEGO® and the Build HAT



Step 1 Introduction

In this project, you will use the Raspberry Pi Build HAT, a LEGO® Technic™ motor encoder and wheel, and the Python Turtle library to make a simple game controller that you can use to play Pong.

> Pong (https://en.wikipedia.org/wiki/Pong) is one of the earliest arcade video games, originally released in 1972 by Atari. It is a table tennis game featuring simple two-dimensional graphics. Players control paddles on each side of the screen, which they use to hit a ball back and forth.

You will:

- Learn how to read the degrees of rotation from LEGO® Technic™ motors
- Learn to draw and move Turtle graphics using LEGO® Technic™ motors
- Learn to detect collisions between graphics using `x` and `y` coordinates

You will need

- A Raspberry Pi computer
- A Raspberry Pi Build HAT
- At least one LEGO® Technic™ motor
- Assortment of LEGO®, including wheels (we used a selection from the LEGO® Education SPIKE™ Prime kit (https://education.lego.com/en-gb/product/spike-prime))
- A small breadboard (optional)
- A buzzer (optional)
- Some breadboard jumper leads (optional)
- A 7.5V power supply with a barrel jack (optional). You can use an official Raspberry Pi power supply for this project, as the motor encoders will not be using any power

Software

- Python 3
- Build HAT Python library

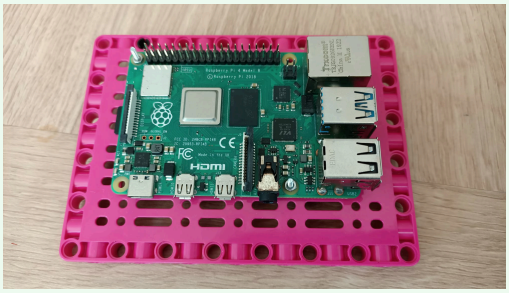
Additional information for educators

You can download the completed project here (https://rpf.io/p/en/lego-game-controller-get).

If you need to print this project, please use the printer-friendly version (https://projects.raspberrypi.org/en/projects/lego-game-controller/print).

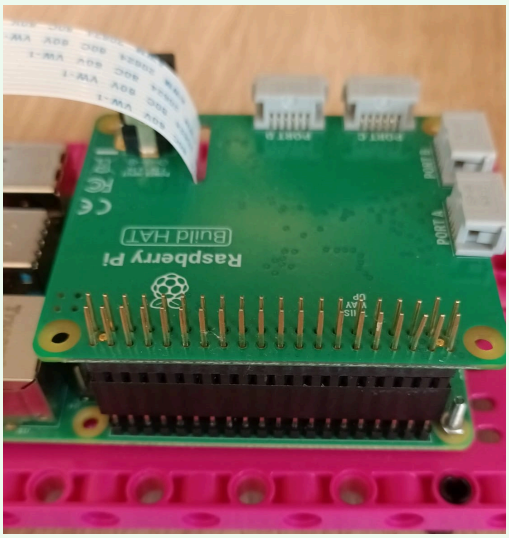
Before you begin, you'll need to have set up your Raspberry Pi computer and attached your Build HAT:

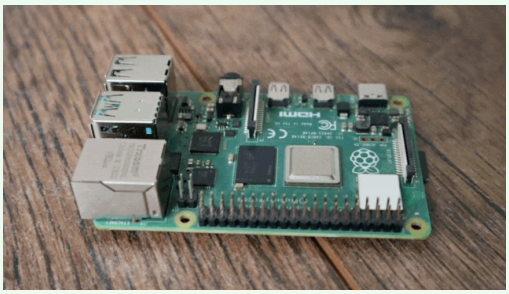
- [ ] Mount your Raspberry Pi on to the LEGO Maker Plate using M2 bolts and nuts, making sure the Raspberry Pi is on the side without the 'edge':



Mounting the Raspberry Pi this way round enables easy access to the ports as well as the SD card slot. The Maker Plate will allow you to connect the Raspberry Pi to the main structure of your dashboard more easily.

Line up the Build HAT with the Raspberry Pi, ensuring you can see the `This way up` label. Make sure all the GPIO pins are covered by the HAT, and press down firmly. (The example uses a stacking header (https://www.adafruit.com/product/2223), which makes the pins longer.)





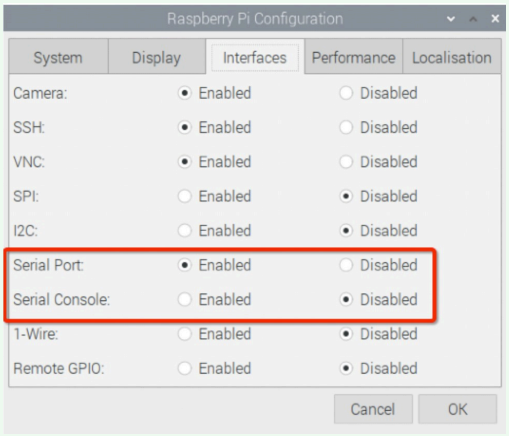
You should now power your Raspberry Pi using the 7.5V barrel jack on the Build HAT, which will allow you to use the motors.

If you have not already done so, set up your Raspberry Pi by following these instructions:

Setting up your Raspberry Pi (https://projects.raspberrypi.org/en/projects/raspberry-pi-setting-up)

Once the Raspberry Pi has booted, open the Raspberry Pi Configuration tool by clicking on the Raspberry Menu button and then selecting “Preferences” and then “Raspberry Pi Configuration”.

Click on the “interfaces” tab and adjust the Serial settings as shown below:



You will also need to install the buildhat python library by following these instructions:

Install the buildhat Python library

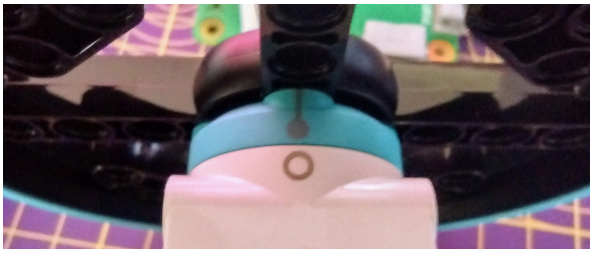
Open a terminal window on your Raspberry Pi by pressing `Ctrl+Alt+T`.

At the prompt type: `sudo pip3 install buildhat`

Press `Enter` and wait for the “installation completed” message.

Step 2 Use LEGO® Spike™ motor encoders

Motor encoders can not only rotate, they can also accurately detect how many degrees they have been rotated.



The LEGO® Spike™ motors all have encoders. If you look at the rotating disk part of the motor, you will see a mark shaped like a lollipop that can be lined up with the 0 mark on the white body of the motor itself. This is the encoder set to zero degrees and any angular movement of the motor shaft can be measured relative to this point.

How motor encoders work

A motor encoder, also called a rotary or shaft encoder, is an electro-mechanical device that allows you to record the angular position or motion of the axle. It normally does this by converting the angular position to an analogue or digital output.

If a motor has an encoder, that means you can very accurately set the position of the axle. It also allows you to use the motor as an input device so that if something changes the position of the axle, this can be registered and used to trigger other actions in a computer program.

Connect a monitor, keyboard, and mouse to your Raspberry Pi device.

Connect your Build HAT to your Raspberry Pi with the printed logo facing up, making sure that you have properly covered all the pins.

Lastly, connect the power; either though the Build HAT barrel jack or the USB-C port on the Raspberry Pi.

Connect a motor to port A on the Build HAT.

Attach a large wheel to the motor using four connector pegs. Turn the wheel so that the lollipop mark is in line with the zero.

Open Thonny from the Raspberry Pi **Programming** menu and click on the **Shell** box at the bottom of the window.

First, import the Build HAT library.

```
from buildhat import Motor
```

Press Enter.

Then, create a motor object that tells Python the motor is connected to port `A`.
Type:

```
motor_left = Motor('A')
```

Press Enter. (There will be a slight delay, be patient!)

Now, you can ask the motor to report its **absolute** position. This will always be between `-180` and `180`.

```
motor_left.get_aposition()
```

Depending on how well you positioned the motor at the start, you should get a value close to `0`.

Move the motor and type the line a second time, and see how the value changes.

You can also keep track of the motor's **relative** position. This is how far it has moved from the time the program starts, so it will increase or decrease by `360` for every turn of the wheel.

```
motor_left.get_position()
```

Move the motor around and check its absolute and relative positions, so that you understand how the values change.

Step 3 React to motor encoder movement

To use the LEGO® Technic™ motors as a controller for a game, you'll need to be able to constantly read their absolute positions.

In the main Thonny window above the shell you can use the commands you already know to find the absolute position of the motor. Then, in a `while True:` loop you can print the value of the position.

game_controller.py

```
from buildhat import Motor
motor_left = Motor('A')

while True:
    print(motor_left.get_aposition())
```

You should see that your program continually prints the position of the motor. If you rotate the motor, these values should change.

There is a better way of doing this though. You only need to read the motor position if it is moved.

Delete the `while True` loop from your program and create this simple function that prints the absolute position of the motor. You will also need to add another import line to use the `pause()` function.

game_controller.py

```
from buildhat import Motor
from signal import pause
motor_left = Motor('A')


def moved_left(motor_speed, motor_pos, motor_apos):
    print(motor_apos)
```

Now set this function to run when the motor's encoder is moved:

game_controller.py

```
from buildhat import Motor
from signal import pause
motor_left = Motor('A')


def moved_left(motor_speed, motor_pos, motor_apos):
    print(motor_apos)

motor_left.when_rotated = moved_left
pause()
```

Run your code and you should see the values printed out in the shell change when the motor is moved.


Save your project

Step 4 Make your Pong screen

Turtle is a drawing and animation library and you can learn more about it with this excellent project (https://projects.raspberrypi.org/en/projects/turtle-race).

First, create a window where the game will be played.

Open a new file in Thonny and add the following code to import the Turtle, time, and Build HAT libraries, and then set up a screen. Run the file and you should see a black window with the title “PONG” open. You don’t need to include the # comments.

pong.py

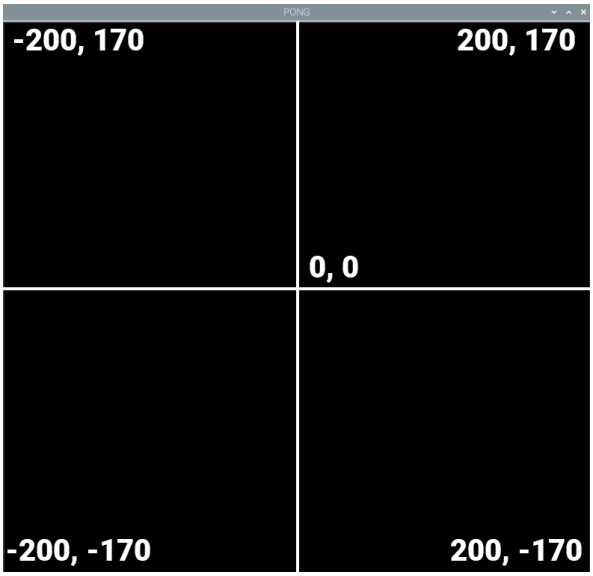
```
from turtle import Screen, Turtle
from time import sleep
from buildhat import Motor

game_area = Screen() #Create a screen
game_area.title("PONG") #Give the screen a title
game_area.bgcolor('black') #Set the background colour
game_area.tracer(0) #Give smoother animations
```

The Turtle library also has a useful way of setting the coordinates for a screen area. Add this line to your program:

pong.py

```
game_area.tracer(0)
game_area.setworldcoordinates(-200, -170, 200, 170)
```

This creates a rectangular window 400 pixels wide and 340 high, with 0 being in the centre.



Now, you need to update your game area, to see the paddle and ball. Add a **game loop** to the bottom of your code, and call the `update()` method.

pong.py

```
10 while True:
11     game_area.update()
```

Run your code and you should see a black window appear.

Next, you can make a ball by using a Turtle that is set to be a white circle. The ball should start in the middle of the screen, and shouldn't draw a line when it moves.

Above your `while True` loop, add the following code:

pong.py

```
10 ball = Turtle()
11 ball.color('white')
12 ball.shape('circle')
13 ball.penup()
14 ball.setpos(0,0)
15
16 while True:
```

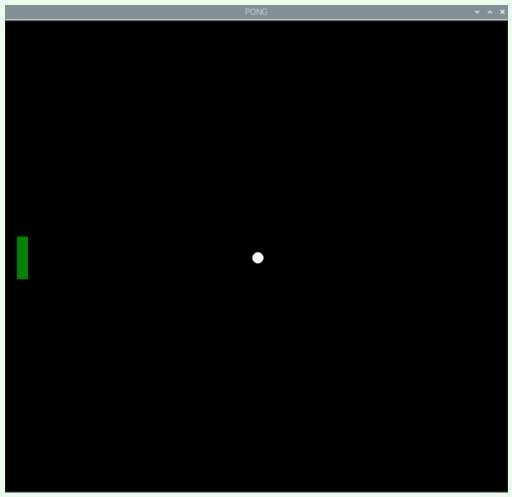
Run your code again. You should see a white ball appear in your window.

Next, you can set up a paddle in the same way. It will be a green rectangle, and positioned on the far left of the screen.

pong.py

```
17 paddle_left = Turtle()
18 paddle_left.color('green')
19 paddle_left.shape('square')
20 paddle_left.shapesize(4, 1, 1)
21 paddle_left.penup()
22 paddle_left.setpos(-190, 0)
```

Run your code and you should see your ball and paddle.



Step 5 Move the ball

The ball is going to bounce around the screen, so two variables are needed to keep track of its speed in both the `x` and `y` dirctions. These numbers can be larger to make the game harder, or smaller to make the game easier.

Add the following code to your program:

pong.py

```
ball.speed_x = 1
ball.speed_y = 1
```

You can check where a Turtle is by using `turtle.xcor()` and `turtle.ycor()` to find the `x` and `y` coordinates, respectively.

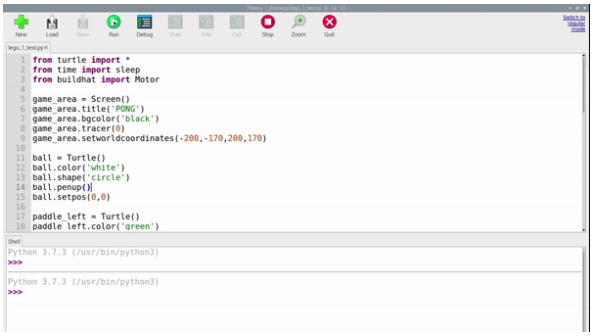
So to make the ball move, you can combine the position and speed.

Add the lines below to your program:

pong.py

```
while True:
    game_area.update()
    ball.setx(ball.xcor() + ball.speed_x)
    ball.sety(ball.ycor() + ball.speed_y)
```

Run the program and see what happens!



The ball should move diagonally upwards towards the top right corner of the game area… and then keep on going! If you want your game to be fast and challenging, you can increase the `speed_x` and `speed_y` values to make the ball move more quickly.

The ball should bounce off the top wall rather than disappear off the screen. To do this, the speed can be reversed, making the ball travel in the opposite direction, if its `y` position is greater than 160.

Add the following code into your game loop and run it.

pong.py

```
while True:
    game_area.update()
    ball.setx(ball.xcor() + ball.speed_x)
    ball.sety(ball.ycor() + ball.speed_y)
    if ball.ycor() > 160:
        ball.speed_y *= -1
```

Run your code again, and the ball should bounce off the top of the screen, but disappear off the right of the screen.

In the same way that the code checks the upper `y` position of the ball, to make it bounce, it can check the right `x` position and the lower `y` position, in your game loop.

Add these checks on the ball's position.

pong.py

```
    if ball.ycor() > 160:
        ball.speed_y *= -1
    if ball.xcor() > 195:
        ball.speed_x *= -1
    if ball.ycor() < -160:
        ball.speed_y *= -1
```

The ball should now bounce around the screen, and fly off the left edge. Next, you will control your paddle to reflect the ball back from the left edge.

 Save your project

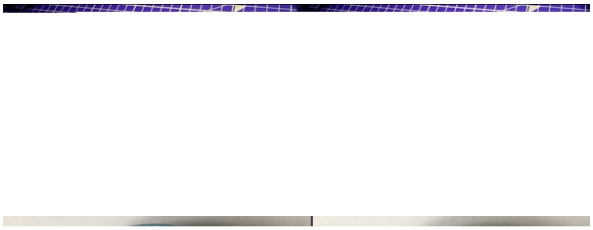
Step 6 Control the paddle

Designing the controls

The LEGO® Spike™ motor is going to be used to control the position of the paddle but you don’t want to be able to make full turns.

A simple way to limit the motion of the wheel is to add a LEGO® element to prevent the wheel turning through a complete rotation.

Line up the encoder marks on your motor using the wheel, like before. Insert a peg or axle as close to level with the markers as possible.



Add a line to create the `motor_left` object after the import line.

pong.py

```
3  from buildhat import Motor
4
5  motor_left = Motor('A')
```

Now a new variable is needed to keep track of the location of the paddle. This will be called `pos_left` and set to `0`.

pong.py

```
26 ball.speed_x = 0.4
27 ball.speed_y = 0.4
28
29 pos_left = 0
```

Create a function for the paddle that will run when the motor encoder moves. Note that it uses a `global` variable so that it can change the value of the `pos_left` variable.

pong.py

```
31 def moved_left(motor_speed, motor_rpos, motor_apos):
32     global pos_left
33     pos_left = motor_apos
```

Now add a single line that will use that function each time the motor is moved. It can be just before your `while` loop.

pong.py

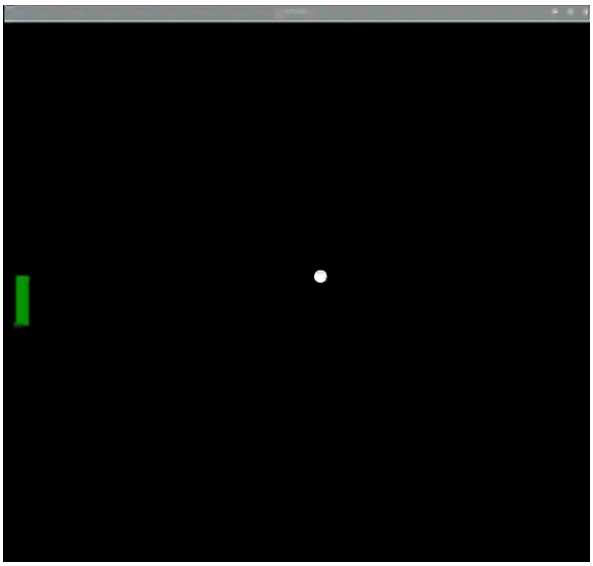
```
35 motor_left.when_rotated = moved_left
```

Then, add a line to the `while True` loop to update the paddle object on the screen to the new position.

pong.py

```
45     if ball.ycor() < -160:
46         ball.speed_y *= -1
47     paddle_left.sety(pos_left)
```

Run your code and then turn the wheel on your motor encoder. You should see your paddle moving up and down the screen.



In case there are errors, your code should currently look like this:

pong.py

```
from turtle import *
from time import sleep
from buildhat import Motor

motor_left = Motor('A')

game_area = Screen()
game_area.title('PONG')
game_area.bgcolor('black')
game_area.tracer(0)
game_area.setworldcoordinates(-200,-170,200,170)

ball = Turtle()
ball.color('white')
ball.shape('circle')
ball.penup()
ball.setpos(0,0)

paddle_left = Turtle()
paddle_left.color('green')
paddle_left.shape("square")
paddle_left.shapesize(4,1,1)
paddle_left.penup()
paddle_left.setpos(-190,0)

ball.speed_x = 0.4
ball.speed_y = 0.4

pos_left = 0


def moved_left(motor_speed, motor_rpos, motor_apos):
    global pos_left
    pos_left = motor_apos


motor_left.when_rotated = moved_left

while True:
    game_area.update()
    ball.setx(ball.xcor() + ball.speed_x)
    ball.sety(ball.ycor() + ball.speed_y)
    if ball.ycor() > 160:
        ball.speed_y *= -1
    if ball.xcor() > 195:
        ball.speed_x *= -1

```

```
    if ball.ycor() < -160:
        ball.speed_y *= -1
paddle_left.sety(pos_left)
```

Save your project

Step 7 Paddle collisions

The game is nearly complete — but first you need to add some extra collision detection that covers the ball hitting the paddle.

Within the `while True` loop, check if the ball's `x` position is within the horizontal area covered by the paddle. Also use an `and` to check the ball's `y` position is in the vertical line in which the paddle moves

pong.py

```
47  paddle_left.sety(pos_left)
48  if (ball.xcor() < -180 and ball.xcor() > -190) and (ball.ycor() <
49  paddle_left.ycor() + 20 and ball.ycor() > paddle_left.ycor() - 20):
50      ball.setx(-180)
        ball.speed_x *= -1
```

Try the program out. You should be able to bounce the ball off your paddle and play a solo game of 'squash'!

Now you have a way of preventing the ball from disappearing off-screen, it's time to think about what happens if you fail to make a save.

For now, let's just reset the ball back to the start.

Add this code within the `while True` loop:

pong.py

```
        ball.speed_x *= -1
    if ball.xcor() < -195: #Left
        ball.hideturtle()
        ball.goto(0,0)
        ball.showturtle()
```

Once you're happy with the various settings, it's time to add in the second paddle.

Using what you've created for the left-hand paddle as a starting point, add a second paddle on the right-hand side of the game area.

First of all, connect a second LEGO® Technic™ motor to the Build HAT (port B) and set it up in the program.

pong.py

```
motor_left = Motor('A')
motor_right = Motor('B')
```

You can copy and paste your code for setting up your left paddle, and change the name and values for your right paddle.

Create your right paddle.

pong

```
paddle_left = Turtle()
paddle_left.color('green')
paddle_left.shape("square")
paddle_left.shapesize(4,1,1)
paddle_left.penup()
paddle_left.setpos(-190,0)

paddle_right = Turtle()
paddle_right.color('blue')
paddle_right.shape("square")
paddle_right.shapesize(4,1,1)
paddle_right.penup()
paddle_right.setpos(190,0)
```

Add a variable for the right paddle position, a function for the paddle, and the line to call the function when the right motor is moved.

pong.py

```
pos_left = 0
pos_right = 0


def moved_left(motor_speed, motor_rpos, motor_apos):
    global pos_left
    pos_left = motor_apos


def moved_right(motor_speed, motor_rpos, motor_apos):
    global pos_right
    pos_right = motor_apos


motor_left.when_rotated = moved_left
motor_right.when_rotated = moved_right
```

Add a line to update the paddle on screen to the `while True` loop:

pong.py

```
    paddle_left.sety(pos_left)
    paddle_right.sety(pos_right)
```

Currently, the ball will bounce off the right-hand wall. Modify the lines of your program that make that happen so that the ball is instead reset to the centre.

Change the condition for the ball's `xcor` so that it resets.

pong.py

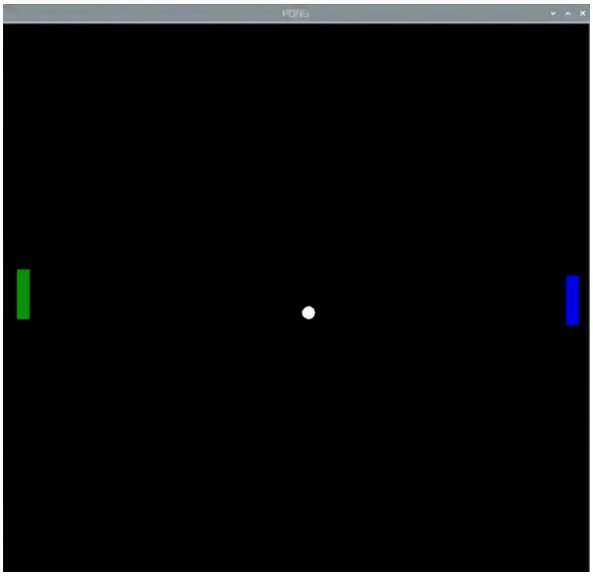
```
60      if ball.xcor() > 195:
61          ball.hideturtle()
62          ball.goto(0,0)
63          ball.showturtle()
```

Now add a similar condition for the right paddle as you did with the left, to handle collisions.

pong.py

```
68      if (ball.xcor() < -180 and ball.xcor() > -190) and (ball.ycor()
69  < paddle_left.ycor() + 20 and ball.ycor() > paddle_left.ycor() -
70  20):
71          ball.setx(-180)
72          ball.speed_x *= -1
73      if (ball.xcor() > 180 and ball.xcor() < 190) and (ball.ycor() <
    paddle_right.ycor() + 20 and ball.ycor() > paddle_right.ycor() -
    20):
            ball.setx(180)
            ball.speed_x *= -1
```

You should now be able to enjoy a basic two-player game of Pong.



Save your project

Step 8 Improve your project

There are few additional features you can add to finish off your game.

Adding a score

Keep track of the score by using two variables (one for each player) and update them whenever a round is lost.

I need a hint

```
    if ball.xcor() > 195: #Right
        ball.hideturtle()
        ball.goto(0,0)
        ball.showturtle()
        score_r+=1
    if ball.xcor() < -195: #Left
        ball.hideturtle()
        ball.goto(0,0)
        ball.showturtle()
        score_l+=1
```

Now you need to display the score on the game area. You can use a fourth Turtle to do this.

Add the following to your program after the creation of the paddle and ball Turtles, but before the `while True` loop.

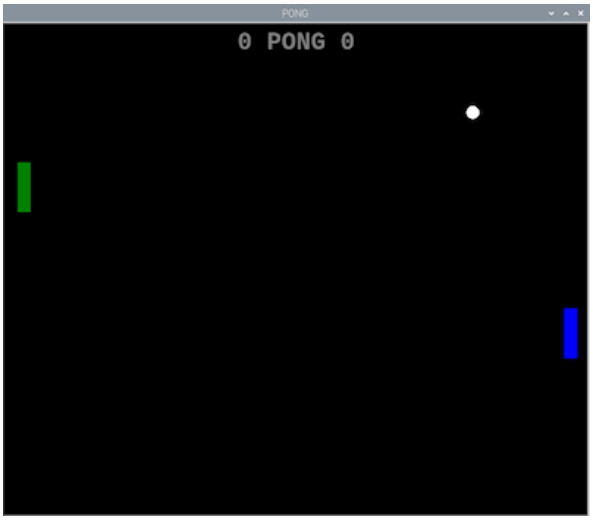
```
writer = Turtle()
writer.hideturtle()
writer.color('grey')
writer.penup()
style = ("Courier",30,'bold')
writer.setposition(0,150)
writer.write(f'{score_l} PONG {score_r}', font=style, align='center')
```

You can look at the documentation for the Turtle library to see what other options there are for how the text is displayed.

If you run your program now, the score and Pong legend should appear, but the scores themselves won't get updated.

Find the two conditionals for each of the scoring situations — when ball is missed by a paddle and disappears to the left or right — and update the score by re-writing the new value.

```
    writer.clear()
    writer.write(f'{score_l} PONG {score_r}', font=style,
align='center')
```



Adding a buzzer

To include some simple sound effects, connect a buzzer to the GPIO pins on the Raspberry Pi.

Connecting a buzzer to the Raspberry Pi

There are two main types of buzzer: **active** and **passive**.

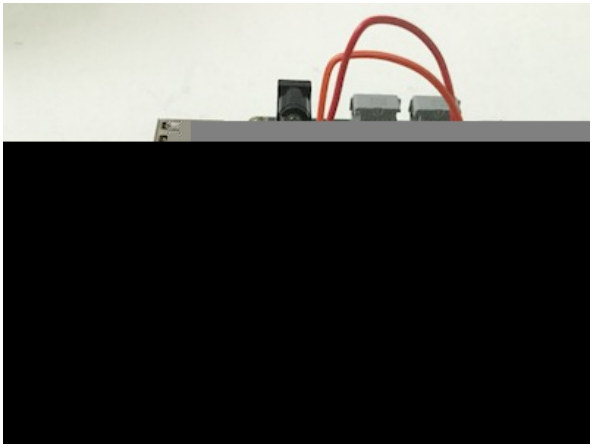
A passive buzzer emits a tone when a voltage is applied across it. It also requires a specific signal to generate a variety of tones. The active buzzers are a lot simpler to use, so these are covered here.

Connecting a buzzer

An active buzzer can be connected just like an LED, but as they are a little more robust, you won't be needing a resistor to protect them.

Set up the circuit as shown below:

Instead of using a breadboard, you could use jumper leads with female sockets at both ends and poke the legs of the buzzer into the socket. Then use some LEGO® elements to mount the buzzer so that it doesn't flop around and become disconnected during frantic gaming sessions.



Now add the `gpiozero` library to the list of imports at the start of you program:

```
from gpiozero import Buzzer
```

Then, make the buzzer available for the program to use by setting which pin you have connected the positive (+) leg to. In this example, we used Pin 17.

```
buzz = Buzzer(17)
```

If you didn't use Pin 17, change the value to reflect the pin your buzzer is connected to.

Now, whenever the paddle and ball make contact, you want the game to play a short tone.

Add this line to each action part of the collision detection `if` conditionals for the ball and paddle:

```
buzz.beep(0.1,0.1,background=True)
```

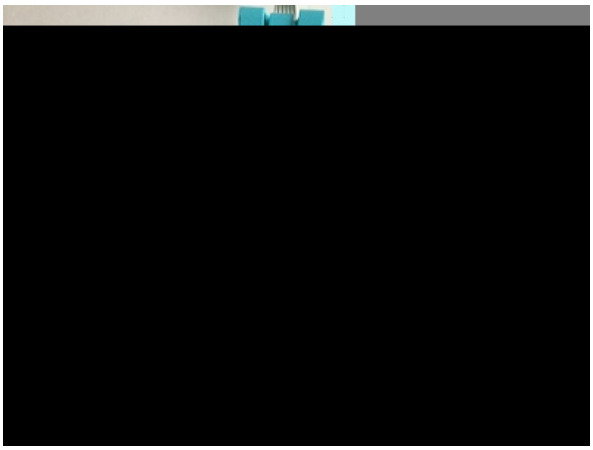
Then add a line to play a longer tone whenever the player misses the ball

```
buzz.beep(0.5,0.5,background=True)
```

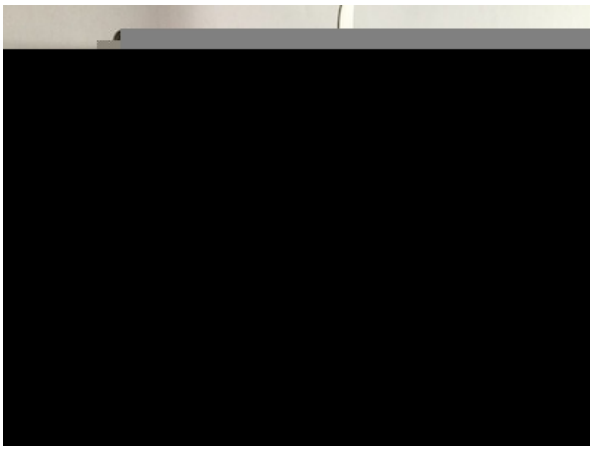
You can read more about the options available with buzzers in the GPIO Zero documentation (https://gpiozero.readthedocs.io/en/stable/api_output.html#buzzer).

Customising your controllers

In your Python Turtle program, you have used different colours for the paddles. You can customise your LEGO® controllers by adding bricks and other LEGO® elements of the same colour.



You could also design a handle for the motor to make it more comfortable to hold.



Your game should now be playable. Have some fun with it before seeing what else you can do next.

Save your project