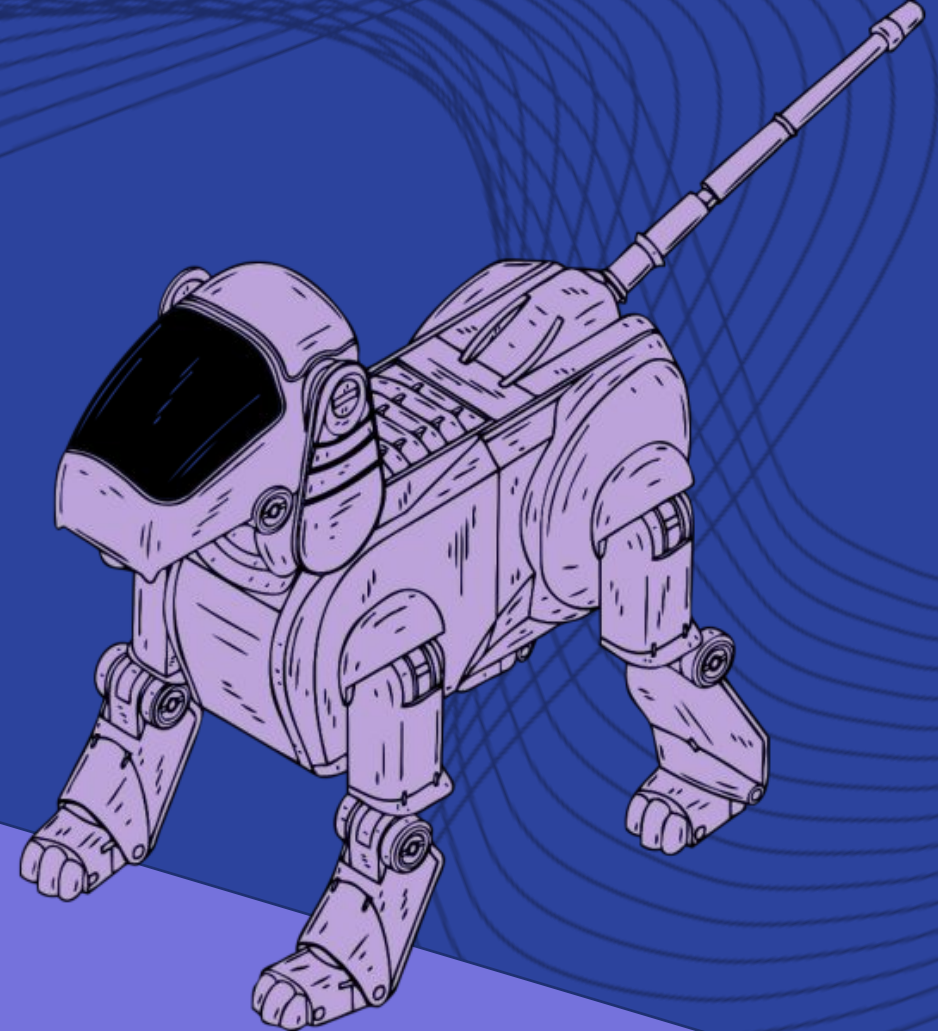


LEGO Pong

Level 3 – Python

At Play

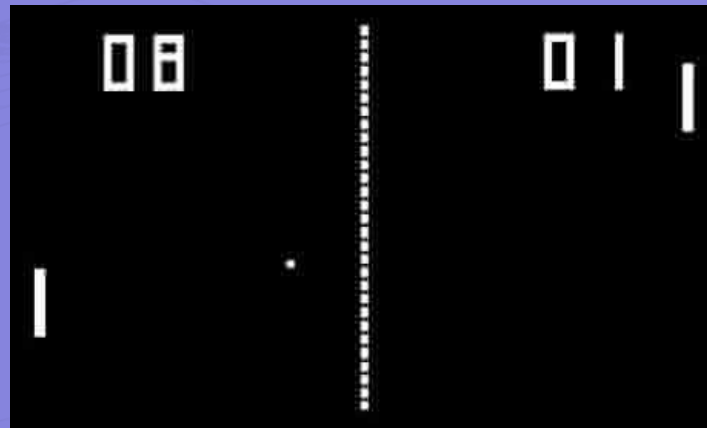
cair
4 YOUTH



Content created by Grace Bennett, Teri Dawkins and Natasha Parbhakar

Introduction


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.



Task

LEGO Pong Game

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.

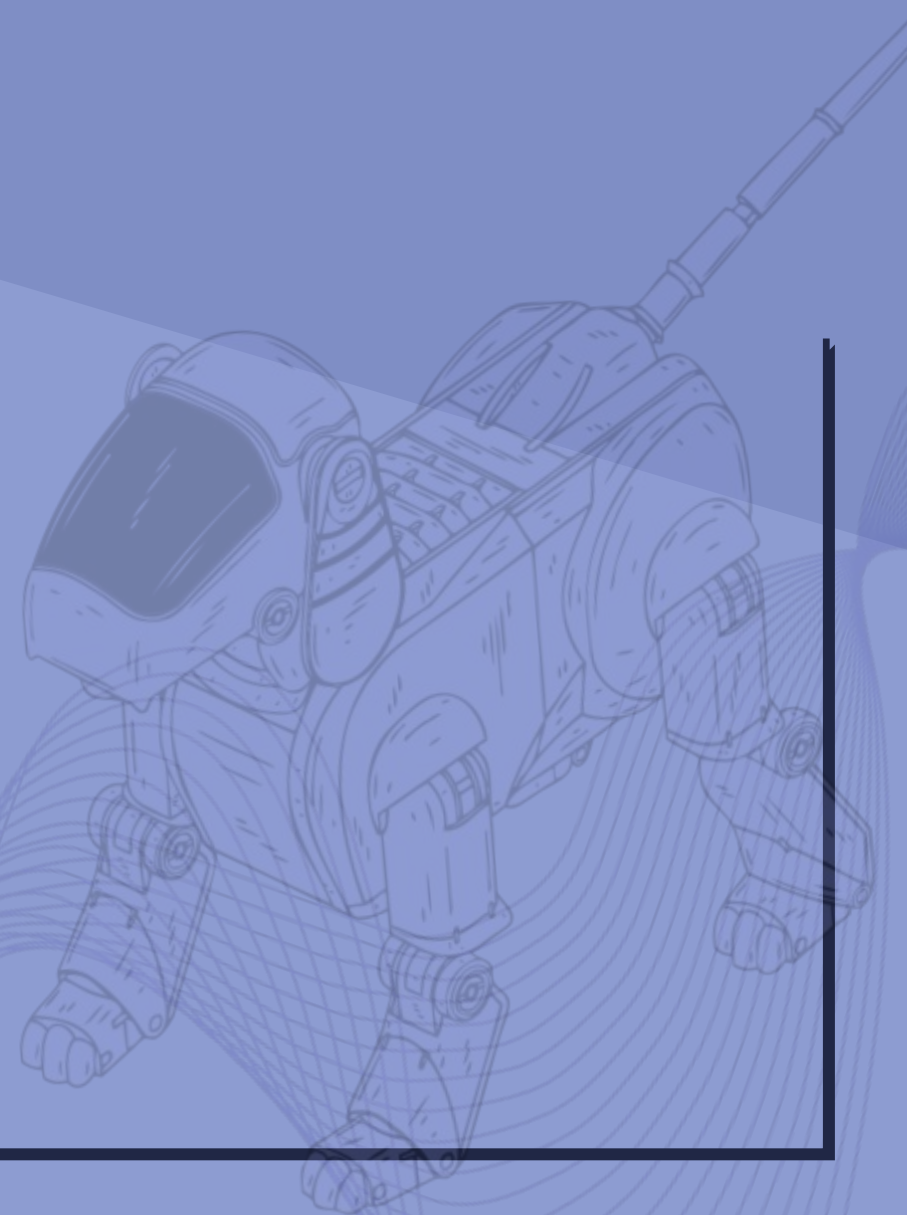




Process

LEGO Pong

- Use LEGO Spike motor encoders
- React to motor encoder movement
- Make your PONG screen
- Move the ball
- Control the paddle
- Code paddle collisions



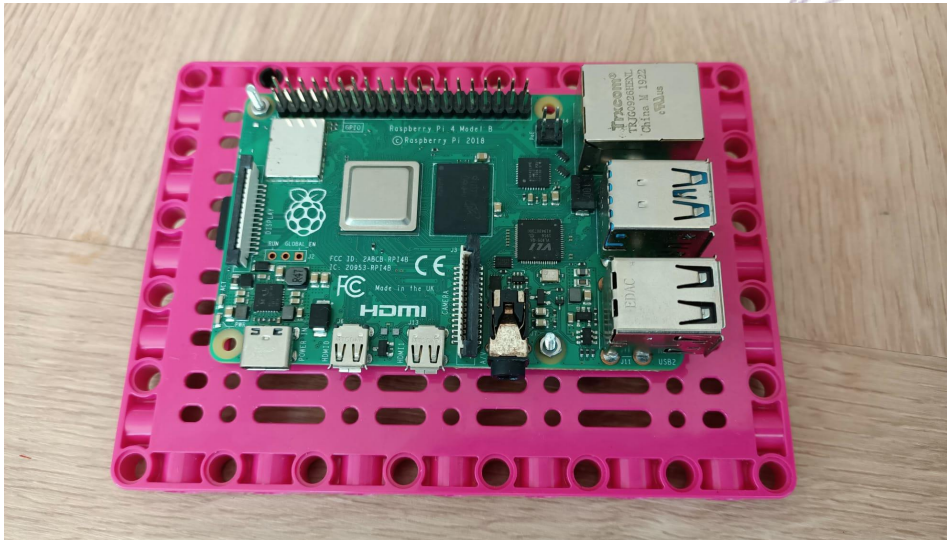
What you will need

LEGO Pong

- A Raspberry Pi
- A Raspberry Pi Build HAT
- At least 1 Lego Technic motor
- Assortment of LEGO®, including wheels (we used a selection from the LEGO® Education SPIKE™ Prime kit)
- 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

Before you begin...

Before you start, you'll need to have set up your Raspberry Pi Computer and attached the Build HAT.



Mount your Raspberry Pi on 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 enables easy access to the ports and SD card.



Line up the Build HAT with the Raspberry Pi, ensuring you can see the **This way up** label. Make sure all GPIO pins are covered by the HAT and press down firmly.

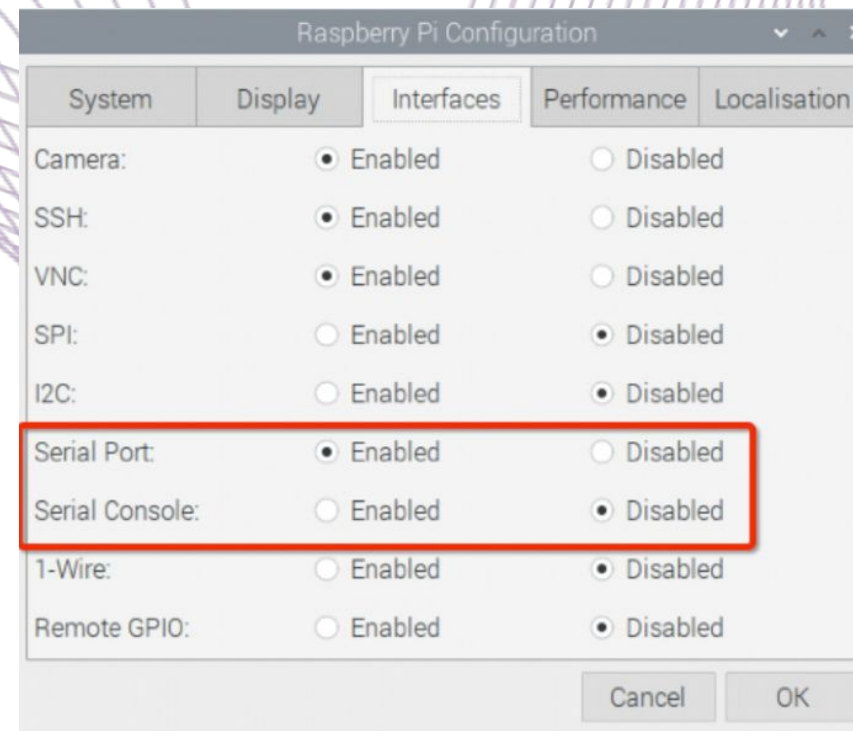
Before you begin...

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

If you need more information about getting started with the Raspberry Pi, click [here](#).

Once the Raspberry Pi has been 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 :



Before you begin...

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

Install the buildhat Python library

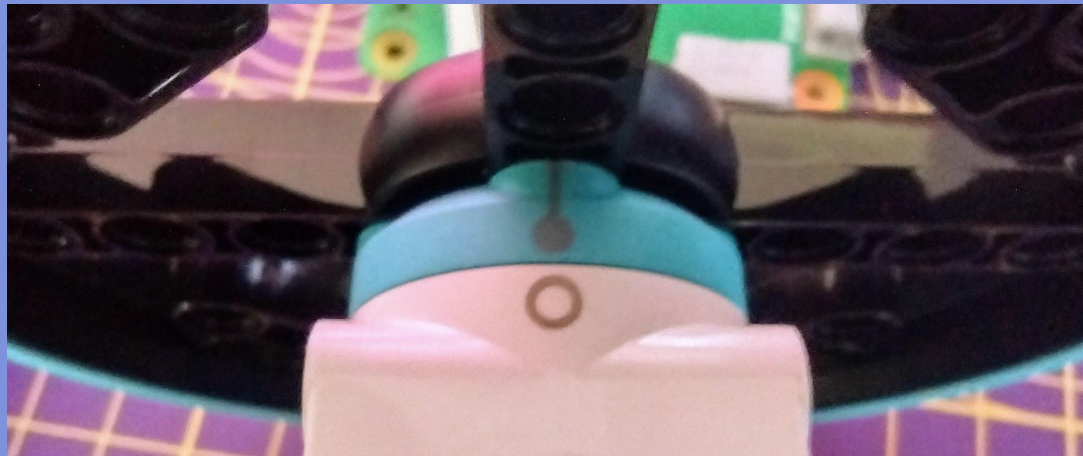
Open a terminal window on your Raspberry Pi by pressing **Ctrl+Alt+T**.

At the prompt type: **`sudo pip3 install buildhat`**

Press **Enter** and wait for the "installation completed" message.

Step 1

Using the 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



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.

Step 2

Using the Lego Spike motor encoders

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

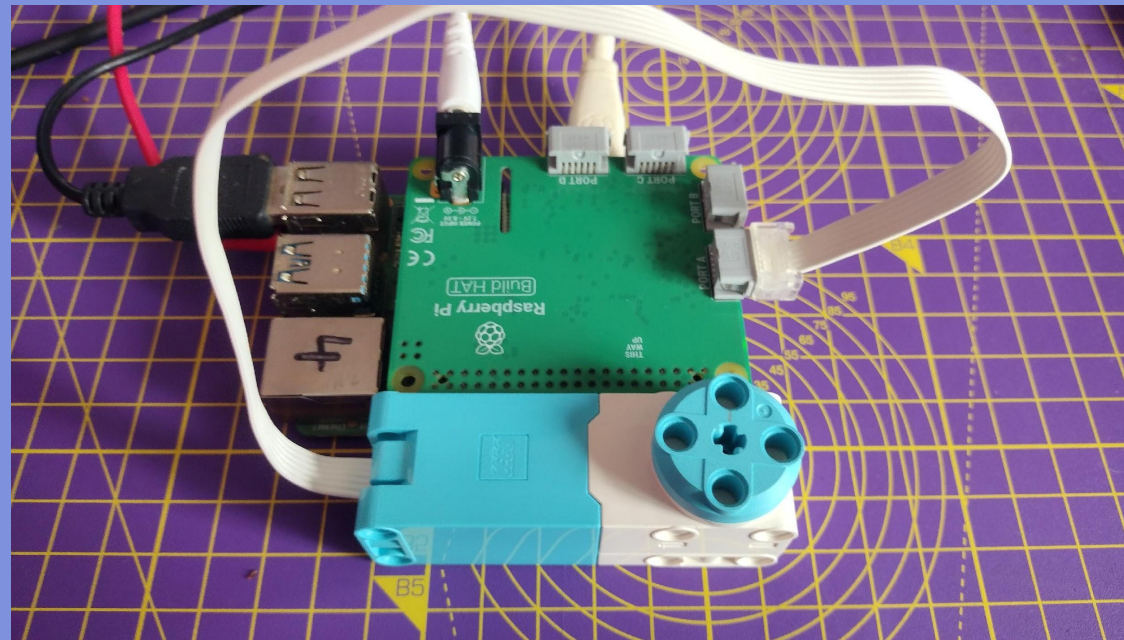
Connect your Build HAT to your Raspberry Pi with the printed logo facing up, making sure that you have properly covered all the pins.

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

Step 3

Using the LEGO spike motor encoders

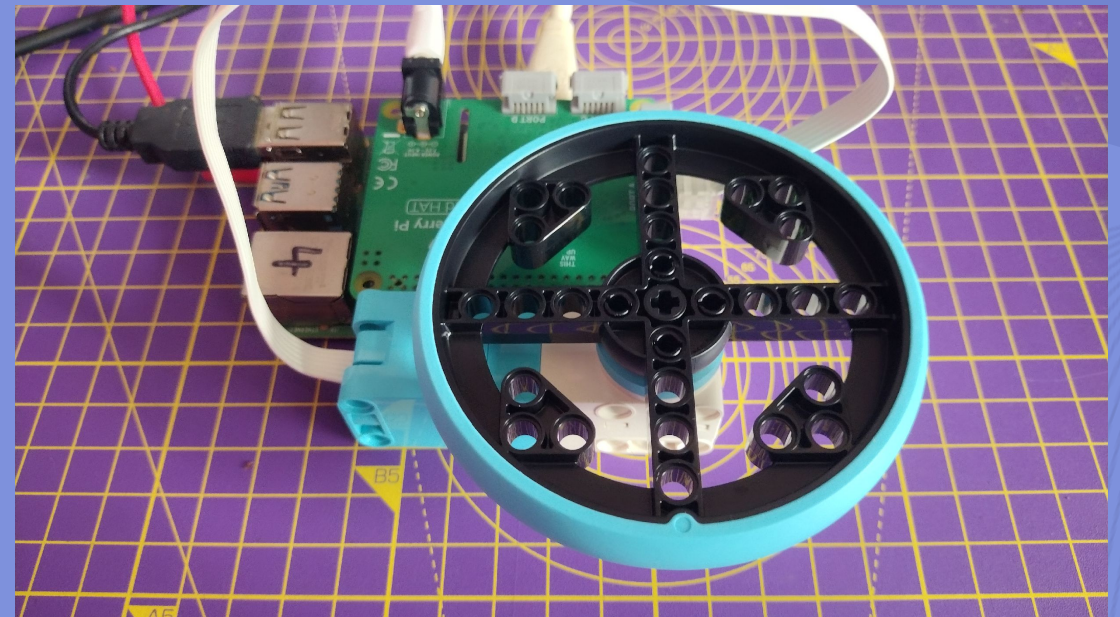
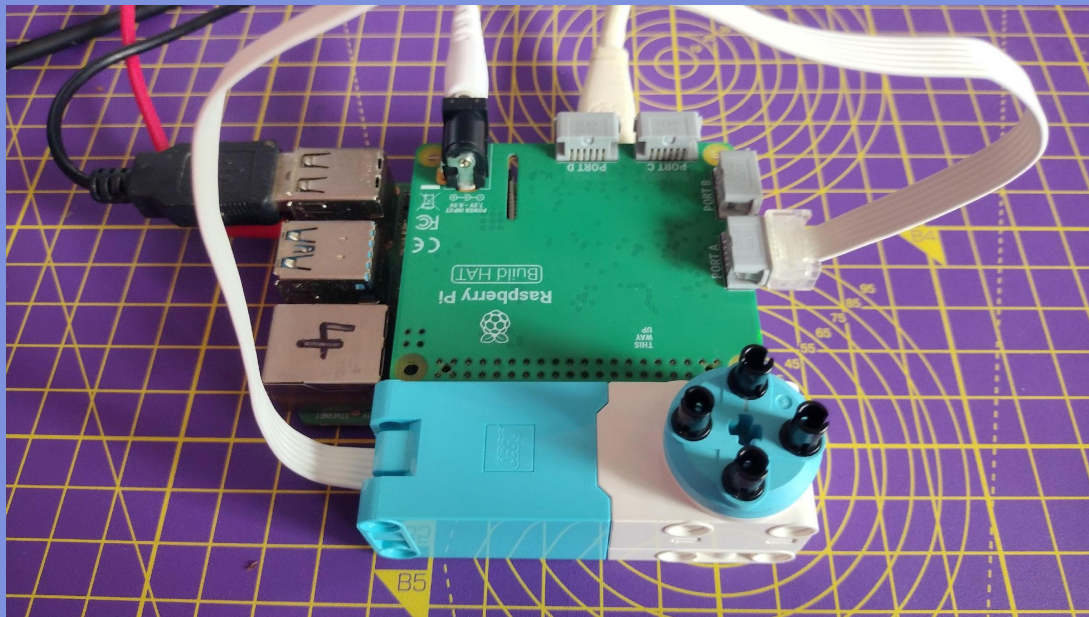
Connect a motor to port A on the Build HAT



Step 4

Using the LEGO spike motor encoders

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



Step 5

Using the LEGO Spike motor encoders

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!)

Step 6

Using the LEGO Spike motor encoders

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 programme 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 7

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.

Make a new file called `game_controller.py`

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.

```
1  from buildhat import Motor
2  motor_left = Motor('A')
3
4  while True:
5      print(motor_left.get_aposition())
```

Step 8

React to motor encoder movement

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 programme 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.

```
1 from buildhat import Motor
2 from signal import pause
3 motor_left = Motor('A')
4
5
6 def moved_left(motor_speed, motor_pos, motor_apos):
7     print(motor_apos)
```

Step 9

React to motor encoder movement

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

```

1  from buildhat import Motor
2  from signal import pause
3  motor_left = Motor('A')
4
5
6  def moved_left(motor_speed, motor_pos, motor_apos):
7      print(motor_apos)
8
9  motor_left.when_rotated = moved_left
10 pause()
```

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

Step 10

Make your pong screen

Turtle is a drawing and animation library - first, create a window where the game will be played.

Make a new file called `pong.py`

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.

```
1 from turtle import Screen, Turtle
2 from time import sleep
3 from buildhat import Motor
4
5 game_area = Screen() #Create a screen
6 game_area.title("PONG") #Give the screen a title
7 game_area.bgcolor('black') #Set the background colour
8 game_area.tracer(0) #Give smoother animations
```

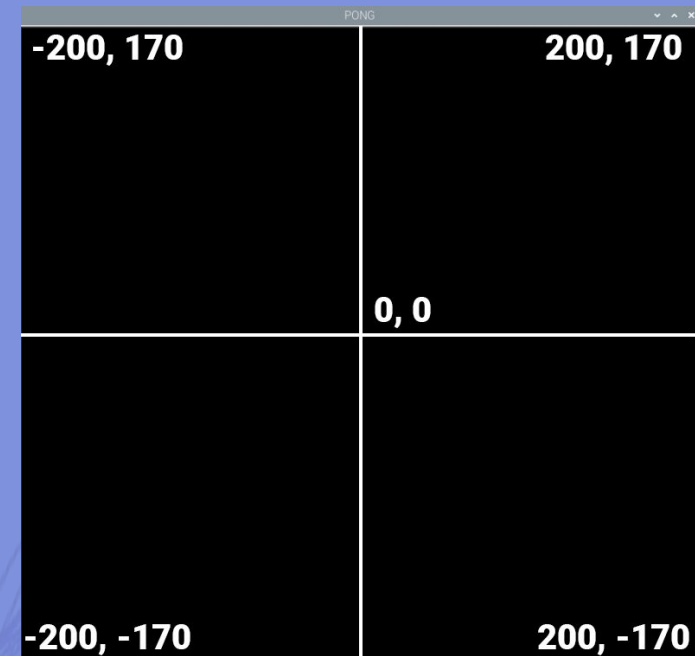

Step 11

Make your pong screen

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

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

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



Step 12

Make your pong screen

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.

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

Run your code and a black window should 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:

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

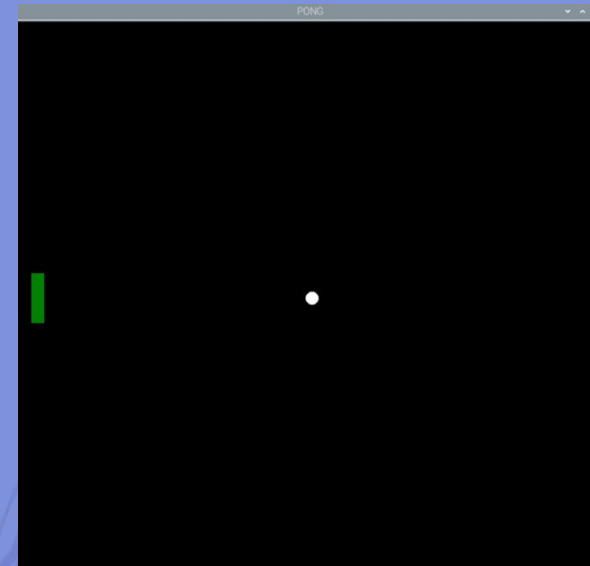
Step 13

Make your pong screen

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.

```
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 14

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 directions. These numbers can be larger to make the game harder, or smaller to make the game easier.

Add the following code to your programme:

```
23 | ball.speed_x = 1
24 | 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.

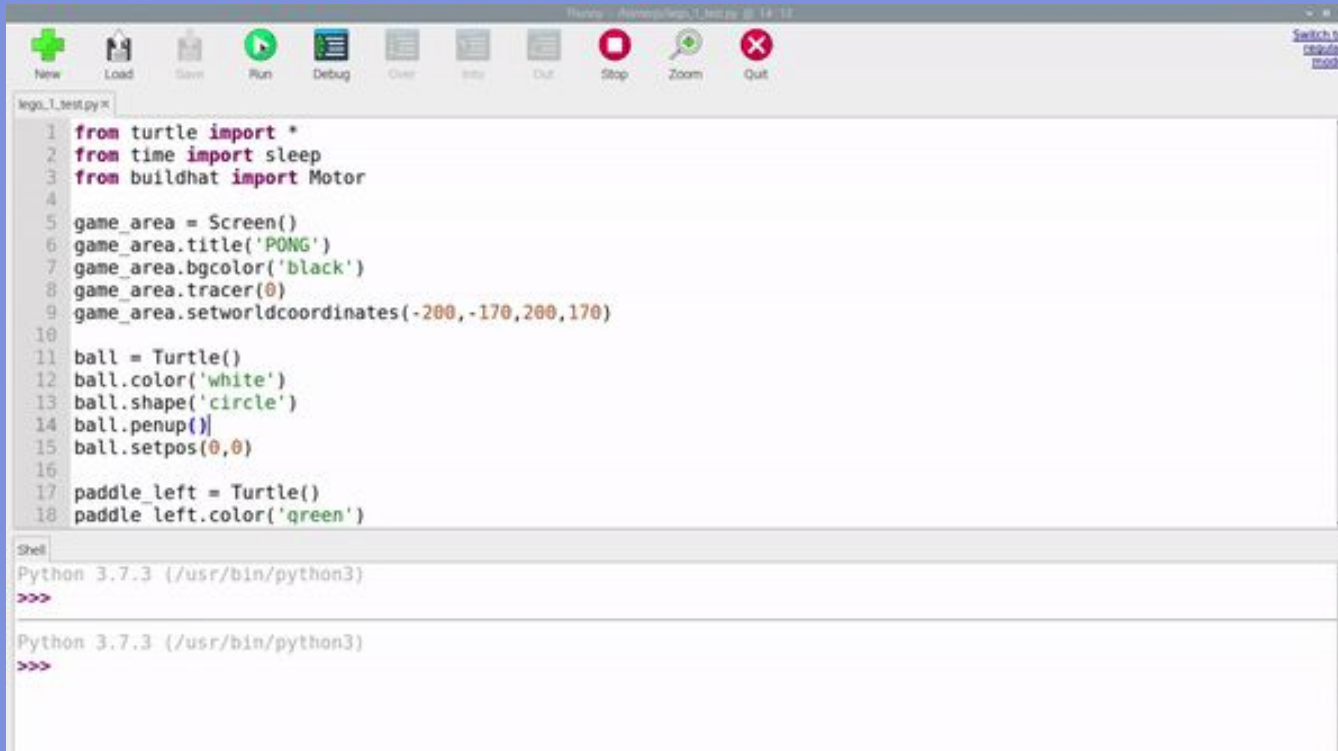
So to make the ball move, you can combine the position and speed.

```
27 | while True:
28 |     game_area.update()
29 |     ball.setx(ball.xcor() + ball.speed_x)
30 |     ball.sety(ball.ycor() + ball.speed_y)
```


Step 15

Move the ball

Run the programme and see what happens!



```

1 from turtle import *
2 from time import sleep
3 from buildhat import Motor
4
5 game_area = Screen()
6 game_area.title('PONG')
7 game_area.bgcolor('black')
8 game_area.tracer(0)
9 game_area.setworldcoordinates(-200,-170,200,170)
10
11 ball = Turtle()
12 ball.color('white')
13 ball.shape('circle')
14 ball.penup()
15 ball.setpos(0,0)
16
17 paddle_left = Turtle()
18 paddle_left.color('green')
  
```

Shell

```

Python 3.7.3 (/usr/bin/python3)
>>>

Python 3.7.3 (/usr/bin/python3)
>>>
  
```

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.

Step 16

Move the ball

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.

```
27 while True:
28     game_area.update()
29     ball.setx(ball.xcor() + ball.speed_x)
30     ball.sety(ball.ycor() + ball.speed_y)
31     if ball.ycor() > 160:
32         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.

Step 17

Move the ball

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.

```
32  if ball.ycor() > 160:
33      ball.speed_y *= -1
34  if ball.xcor() > 195:
35      ball.speed_x *= -1
36  if ball.ycor() < -160:
37      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.

Step 18

Control the paddle

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.



Step 19

Control the paddle

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

```
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.

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

Step 20

Control the paddle

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.

```
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.

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

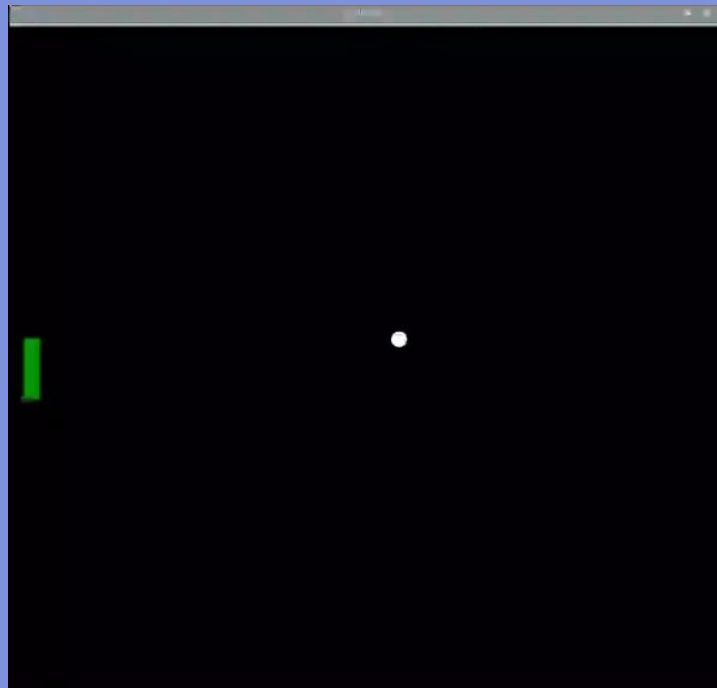
Then, add a line to the while True loop to update the paddle object on the screen to the new position.

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

Step 20

Control the paddle

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



What your code should look like...

```

1  from turtle import *
2  from time import sleep
3  from buildhat import Motor
4
5  motor_left = Motor('A')
6
7  game_area = Screen()
8  game_area.title('PONG')
9  game_area.bgcolor('black')
10 game_area.tracer(0)
11 game_area.setworldcoordinates(-200,-170,200,170)
12
13 ball = Turtle()
14 ball.color('white')
15 ball.shape('circle')
16 ball.penup()
17 ball.setpos(0,0)
18
19 paddle_left = Turtle()
20 paddle_left.color('green')
21 paddle_left.shape("square")
22 paddle_left.shapesize(4,1,1)
23 paddle_left.penup()
24 paddle_left.setpos(-190,0)
25

```

```

25
26 ball.speed_x = 0.4
27 ball.speed_y = 0.4
28
29 pos_left = 0
30
31
32 def moved_left(motor_speed, motor_rpos, motor_apos):
33     global pos_left
34     pos_left = motor_apos
35
36
37 motor_left.when_rotated = moved_left
38
39 while True:
40     game_area.update()
41     ball.setx(ball.xcor() + ball.speed_x)
42     ball.sety(ball.ycor() + ball.speed_y)
43     if ball.ycor() > 160:
44         ball.speed_y *= -1
45     if ball.xcor() > 195:
46         ball.speed_x *= -1
47     if ball.ycor() < -160:
48         ball.speed_y *= -1
49     paddle_left.sety(pos_left)

```


Step 21

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.

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

Step 22

Paddle Collisions

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:

```
52 | ball.speed_x *= -1
53 | if ball.xcor() < -195: #Left
54 |     ball.hideturtle()
55 |     ball.goto(0,0)
56 |     ball.showturtle()
```

Step 23

Setting up the second paddle

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 programme.

```
5 | motor_left = Motor('A')
6 | motor_right = Motor('B')
```

Step 24

Setting up the second paddle

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.

```

20 paddle_left = Turtle()
21 paddle_left.color('green')
22 paddle_left.shape("square")
23 paddle_left.shapesize(4,1,1)
24 paddle_left.penup()
25 paddle_left.setpos(-190,0)
26
27 paddle_right = Turtle()
28 paddle_right.color('blue')
29 paddle_right.shape("square")
30 paddle_right.shapesize(4,1,1)
31 paddle_right.penup()
32 paddle_right.setpos(190,0)

```


Step 25

Setting up the second paddle

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.

```

37 pos_left = 0
38 pos_right = 0
39
40
41 def moved_left(motor_speed, motor_rpos, motor_apos):
42     global pos_left
43     pos_left = motor_apos
44
45
46 def moved_right(motor_speed, motor_rpos, motor_apos):
47     global pos_right
48     pos_right = motor_apos
49
50
51 motor_left.when_rotated = moved_left
52 motor_right.when_rotated = moved_right

```

Step 26

Setting up the second paddle

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

```
64 | paddle_left.sety(pos_left)
65 | 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.

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

Step 27

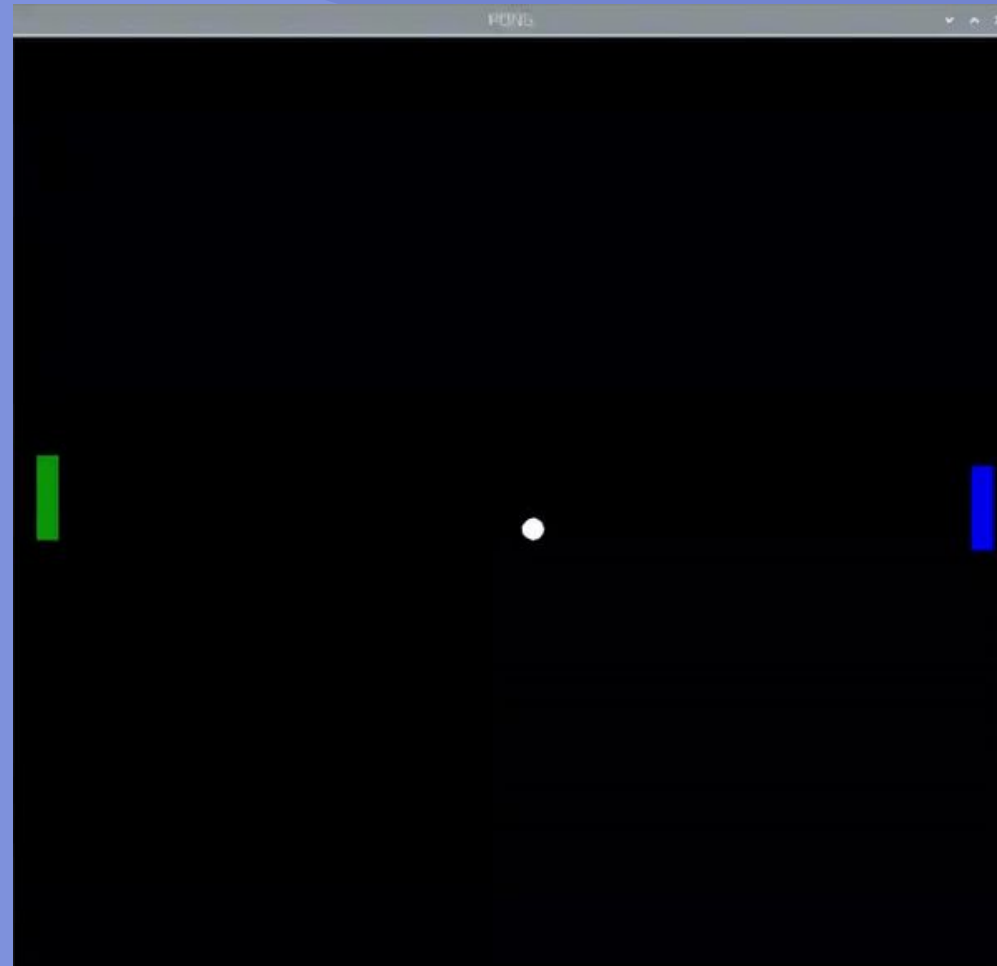
Setting up the second paddle

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

```
if (ball.xcor() < -180 and ball.xcor() > -190) and (ball.ycor() < paddle_left.ycor() + 20 and ball.ycor() > paddle_left.ycor() - 20):
    ball.setx(-180)
    ball.speed_x *= -1

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!



Conclusion

Learning outcomes

- ✓ 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.

If you are interested in ways you might improve your project, find more about how you can do that [here](#).

Congratulations!
You have completed the project

