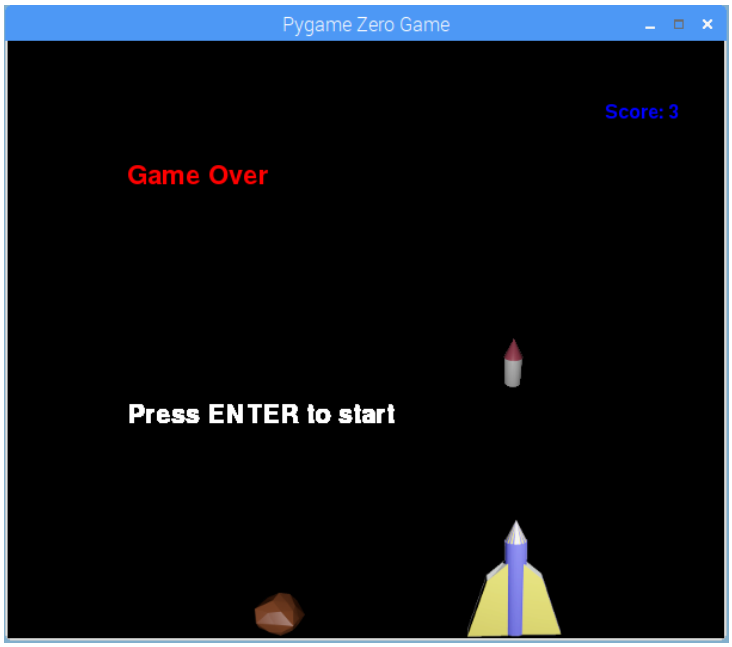
Space Asteroids – Pygame Zero

Physical computing and game programming with the Raspberry Pi

In this worksheet you will learn how to create a computer game including your own interactive game controller made with electronic components.

The game is a combination of the two classic games *Space Invaders* and *Asteroids*. Asteroids are falling towards the earth and need to be stopped using a missile launcher than fires straight up at the incoming asteroids.

A screen-shot of the game is shown below:



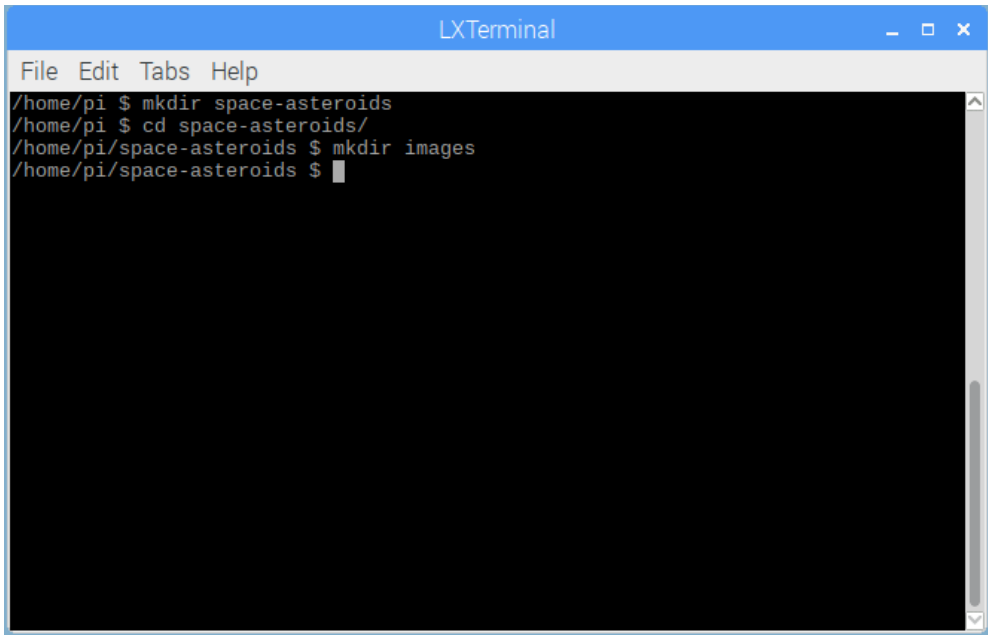
The game uses pygame zero for the graphics and gpio zero for interfacing with the electronics. It also uses three images which were created using Blender.

You will need:

Raspberry Pi (B+, Pi2 or Pi3)
Breadboard
3 x Push-to-make PCB switches
3 x 220Ω resistor
LED
Male-to-female and male-to-male jumper leads

Setting up the basics

To get started then I suggest creating a new directory called space-asteroids and then a directory underneath to hold the images which needs to be called images.



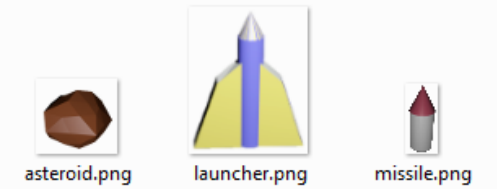
Then download the three images from www.penguintutor.com/space-asteroids and save them in the space-asteroids directory.

Start the Thonny Python IDE from the Programming menu.

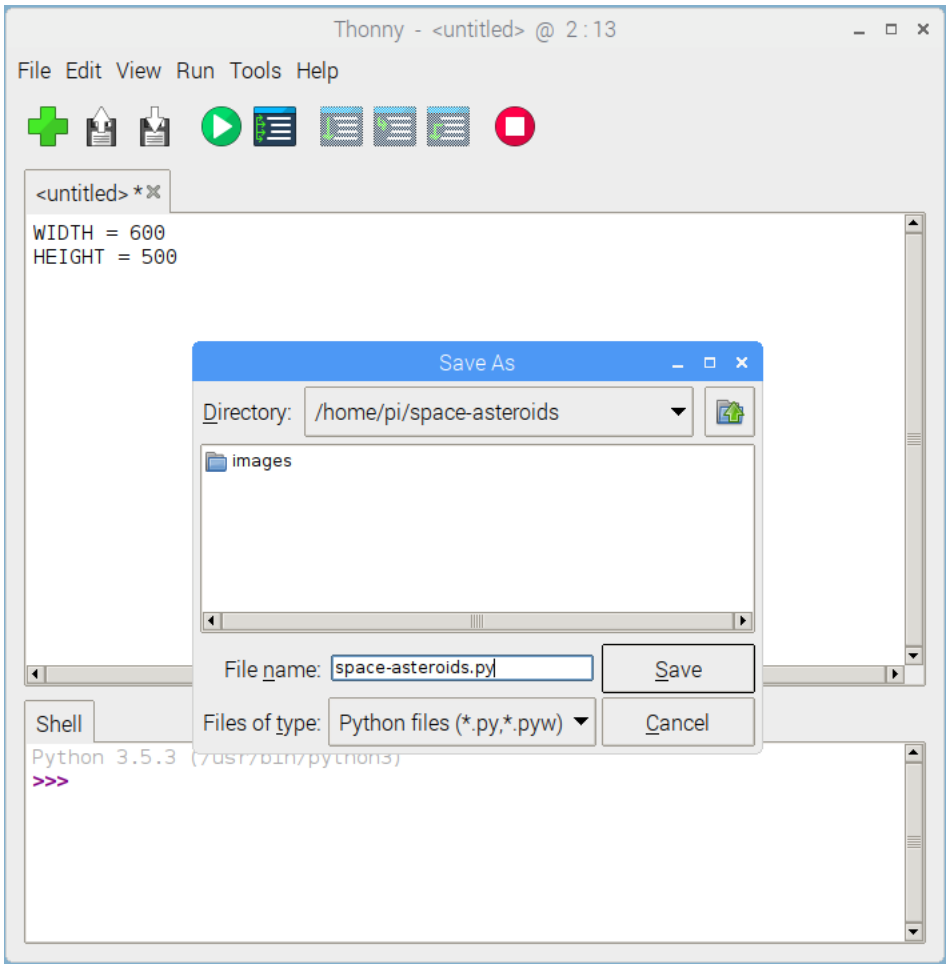
Add the following lines

```
WIDTH = 600
HEIGHT = 500
```

Creating the Images
The images were all created using the 3D modelling tool Blender. You can find details about how to make these images at:
www.penguintutor.com/space-asteroids



Then save as space-asteroids.py in the space-asteroids folder.



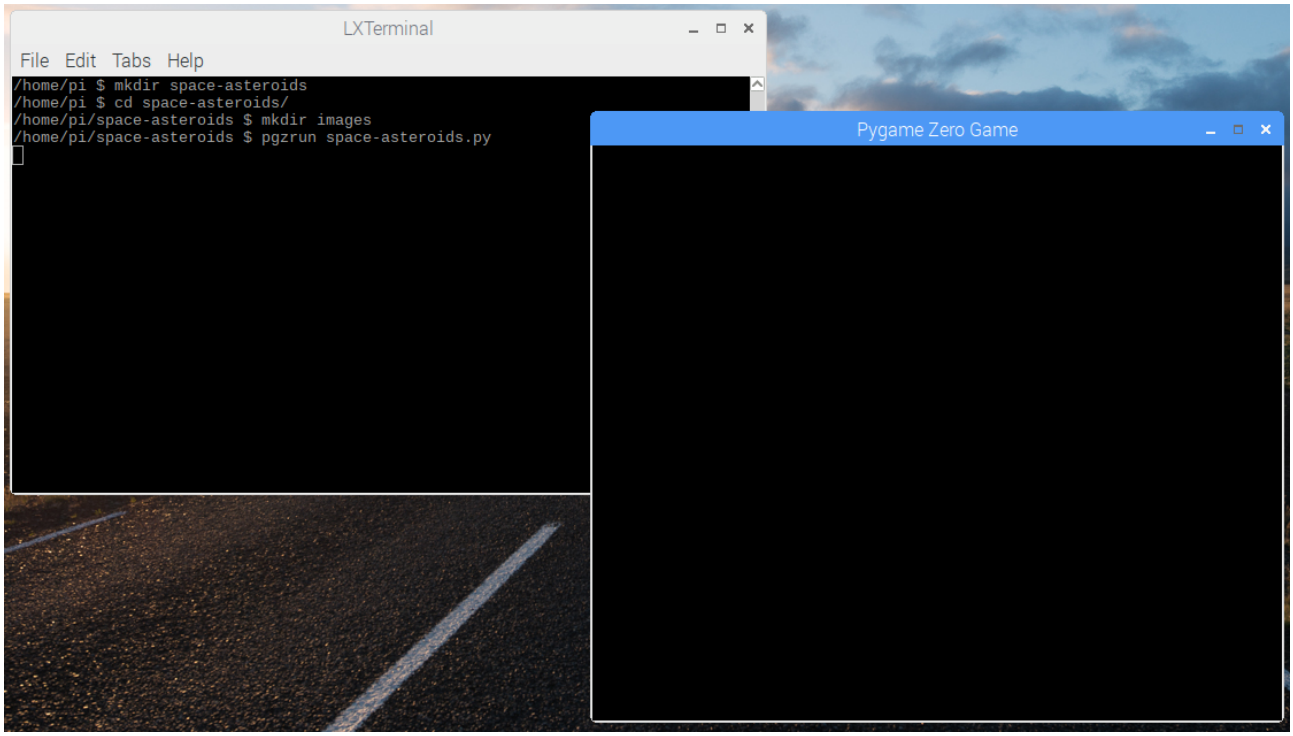
Editor for Pygame Zero

The latest version of the mu editor includes support for Pygame Zero. See https://codewith.mu/ for more details.

At the time of writing this needs to be installed separately, so is not used in this worksheet.

Unfortunately it's not possible to run the code directly from within Thonny, instead you need to go to back to the terminal and run

```
pgzrun space-asteroids.py
```



We haven't added any code yet, but those two lines defining the width and height are enough to create a game window with a blank background. You can close that window for now, but remember whenever you want to run the code go back to the terminal and use pgzrun to launch it.

Creating the launcher

The first thing you will create is a launcher which we can use to defend against the incoming asteroids. There is an image file for this, which looks like a rocket, called launcher.png. This can be moved across the screen using keys or some switches that you will connect to the Raspberry Pi.

Create a launcher by adding the following code after the existing code.

```
# The Launcher
launcher = Actor('launcher')
launcher.pos = int(WIDTH/2),HEIGHT-50
launcher_speed = 2
```

This creates the launcher which is an instance of an Actor (also known as a sprite), and places the launcher in the middle, bottom of the screen. It also defines the speed that the launcher can move across the screen.

Next add two functions, called draw and update. These are both a standard feature of Pygame Zero. The draw function is used to draw items on the screen and the update function is normally used for checking for updates such as key presses. Both functions are run regularly, approximately every 60 seconds.

You will add to these functions as the game is created, for now add the following details:

```
def draw():
    screen.clear()
    launcher.draw()

def update():
    # Keyboard movement
    if keyboard.right:
        move_launcher(launcher_speed)
    if keyboard.left:
        move_launcher(-1 * launcher_speed)
```

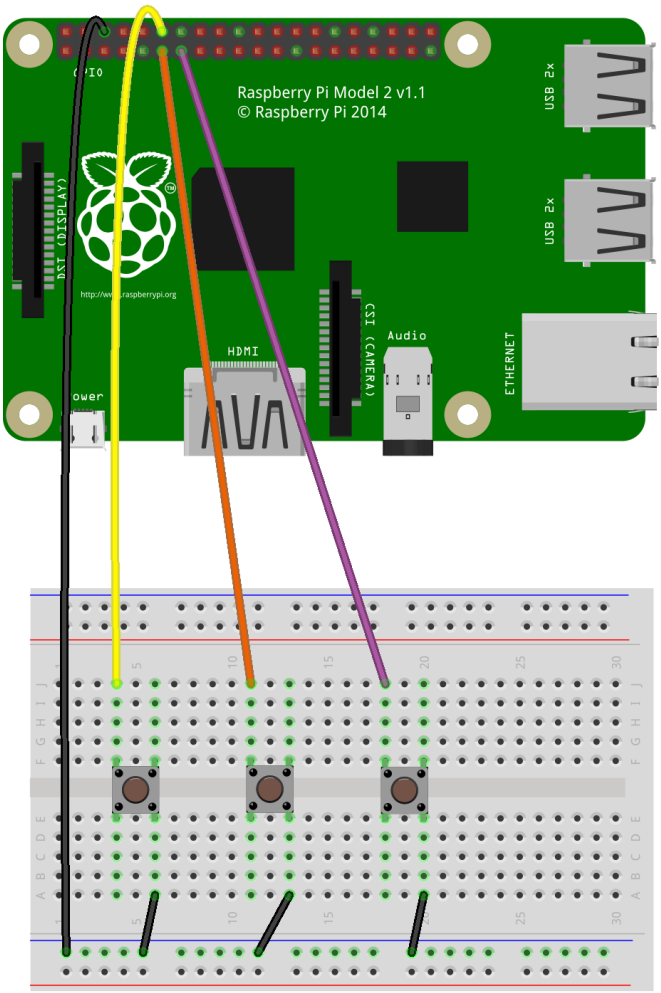
You will also need to add the function that moves the launcher from side to side. This needs to include a check to make sure the launcher does not fall off the end of the screen.

```
# Move launcher allowing, prevent overrunning
def move_launcher(distance):
    newpos = launcher.x + distance
    if (newpos < 50):
        newpos = 50
    elif (newpos > WIDTH-50):
        newpos = WIDTH-50
    launcher.x = newpos
```

After saving the file you should be able to run it (using pgzrun) and move the launcher side-to-side using the cursor keys.

Adding electronic inputs

For the inputs you should connect three switches. The left and right switches will be used for moving the launcher and the centre switch for firing a missile (which you will add later). Wire up the switches to the breadboard as shown in the diagram below. Note that the colours of the wires does not matter as long as the wires connect to the correct positions.



When configured the ports of the Raspberry Pi will be connected through a pull-up resistor, so will normally be set as HIGH. When the button is pressed then it connects the port down to the ground (0v) pin and so changes the value to LOW.
This can be configured by adding some code using GPIO Zero. First add an import statement at the first line which imports GPIO Zero.

```
from gpiozero import Button, LED
```

Next add some constants to define which button relates to which PIN of the GPIO and set the ports as input buttons. The following code should be entered after the HEIGHT entry. Note that these are the logical pin numbers rather than their physical positions.

```
PIN_LEFT = 18
PIN_MISSILE = 17
PIN_RIGHT = 27
button_left = Button(PIN_LEFT)
button_missile = Button(PIN_MISSILE)
button_right = Button(PIN_RIGHT)
```

Now change the update function as below:

```
def update():
    # Keyboard movement
    if keyboard.right or button_right.is_pressed:
        move_launcher(launcher_speed)
    if keyboard.left or button_left.is_pressed:
        move_launcher(-1 * launcher_speed)
```

Running the program again should now allow the launcher to be moved by using either the left and right buttons on the breadboard, or the left and right arrow keys. If it doesn't work at this stage then you need to make sure you have the correct buttons which should be momentary push-to-make buttons and that the wires connect to opposite sides of the switch (normally diagonally opposite pins).

Adding the asteroids

So far you have created a launcher that can move left and right. Now you need something to target. There is an asteroid image that can be used added. This will be displayed at a random position across the top of the screen and then fall to the earth.

First you need to import the random module, which can be added straight after the “from gpiozero ...” line:

```
from gpiozero import Button, LED
import random
```

Next create an instance of the asteroid after the launcher:

```
# The Launcher
...
launcher_speed = 2

# The Asteroid
ASTEROID_SIZE = 50,46
asteroid = Actor('asteroid')
asteroid.pos = (random.randint(ASTEROID_SIZE[0]/2, WIDTH-(ASTEROID_S
IZE[0]/2)), ASTEROID_SIZE[1])
asteroid_speed = 1
```

Then add the following to the update function:

```
    ...
    launcher.draw()
    # Each draw loop move asteroid down 1 position
    move_asteroid()
    asteroid.draw()
```

Then add the move_asteroid function at the bottom of the file:

```
# Move asteroid position and check for hits
def move_asteroid():
    global game_status
    # Move position down
    asteroid.pos = (asteroid.x, asteroid.y+asteroid_speed)
```

If you run the program now then an asteroid will fall down, but not do anything when in reaches the bottom of the screen. In fact it will keep on going down well beyond the bottom of the screen. We will fix that later when we also add the code for firing missiles.

Firing missiles and making a game

In this next part you will build onto what you have achieved so far, adding the ability to fire a missile to defend against the asteroids. This will make the code into a game, complete with collision detection and scoring.

There is quite a bit that needs to be added here, but I'll explain some of the parts that are different.

Start by adding the following code before the draw function, after the code that was used to create the asteroid:

```
# The missile
MISSILE_SIZE = 20,43
missile = Actor('missile')
# initially set missile position in top corner
missile.pos = 0,0
missile_speed = 10
# Status 0 = not fired, 1 = fired
missile_status = 0

# Status 0 = stop, 1 = playing game
game_status = 0
game_score = 0
```

The setup for the missile is similar to what you've already seen when adding the rocket launcher, but it has added a new variable called missile_status. This will be used to determine if the missile is shown (missile_status = 1) or if it should be hidden (missile_status = 0).

The code also includes some global variables to keep track of the game. This includes game_status (which allows us to determine whether the game is in progress, or if it is waiting to start) and game_score which is a variable to keep track of the score.

You will need to make quite a few changes to the draw function so that it shows a different display depending upon whether the game is in progress or not. Also note that much of the previous code has now been indented to come inside the appropriate if clause.

```
def draw():
    if game_status == 0:
        screen.draw.text("Press ENTER to start", (100, 300),
color="white", fontsize=32)
    if game_status == 1:
        screen.clear()
        # display score
        screen.draw.text("Score: " + str(game_score), (WIDTH-100, 50),
color="blue", fontsize=24)
        launcher.draw()
        # Each draw loop move asteroid down 1 position
        move_asteroid()
        asteroid.draw()
        # If missile launched move missile up and show
        if missile_status > 0 :
            move_missile()
            missile.draw()
        # Detect if missile / asteroid has hit etc
        detect_hits()
```

This works by looking at the game_status. If the game_status is 0 (not in progress) then it just shows text saying “Press ENTER to start”. There is no screen.clear() before this, so it will continue to show the last display including the previous score.

The rest of the function is called when the game_status is 1 (game in progress) and it adds drawing of the missile, but only if the missile_status is 1 (has been fired). There is also a detect_hits function called which you will add shortly to handle the situation where an asteroid is hit by the missile, or where an asteroid hits the ground.

The update function also needs a number of changes to handle the different game status and to add the firing of the missile. The entire function is shown below:

```
def update():
    global game_status, game_score, missile_status, asteroid_speed
    if game_status == 0:
        if keyboard.RETURN:
            game_status = 1
            asteroid.pos = (random.randint(ASTEROID_SIZE[0]/2, WIDTH-
(ASTEROID_SIZE[0]/2)), ASTEROID_SIZE[1])
            game_score = 0
            asteroid_speed = 1
    elif game_status == 1:
        # Keyboard movement
        if keyboard.right or button_right.is_pressed :
            move_launcher(launcher_speed)
        if keyboard.left or button_left.is_pressed :
            move_launcher(-1 * launcher_speed)
        # If missile not fired then it can be fired
        if missile_status == 0:
            if keyboard.space or button_missile.is_pressed :
                # set position to location of launcher
                missile.pos = (launcher.x, HEIGHT - 50)
                # set it fired
                missile_status = 1
```

You will also see that there are a number of global variables that are updated by this function which are used to allow updates to the game status and score, the status of the missile, and to the asteroid speed (so that it can be increased to make it more difficult).

Next add the following two functions to the bottom of the file to handle the movement of the missile and to detect whether the asteroid has been hit, or has hit the ground.

```
# If missile launched then move it up appropriate distance
def move_missile():
    global missile_status
    # Only move / draw if missile is fired
    if missile_status > 0:
        missile.pos = (missile.x, missile.y - missile_speed)
        # Near top and not collided so stop missile
        if missile.y < 10:
            missile_status = 0

# Check for hits between asteroid and ground / missile
def detect_hits():
    global game_status, game_score, missile_status, asteroid_speed
    # Check if we have hit the ground (game over)
    if asteroid.y >= HEIGHT-(ASTEROID_SIZE[1]/2):
        screen.draw.text("Game Over", (100, 100), color="red", ▓
fontsize=32)
        # Stop the game
        game_status = 0
    # check if missile has hit asteroid
    if missile_status == 1 and asteroid.collidepoint(missile.pos):
        # When hit asteroid add point and start new
        hit_asteroid()
        missile_status = 0
        # increase asteroid_speed
        asteroid_speed += 1
        game_score += 1

# When hit reset the asteroid position
def hit_asteroid():
    # Set new position
    asteroid.pos = (random.randint(ASTEROID_SIZE[0]/2, WIDTH ▓
(ASTEROID_SIZE[0]/2)), ASTEROID_SIZE[1])
```

(note that the ▓ on some lines indicates that the line is wrapped to the next line, do not include that character or insert a new line).
You now have a working game, which can be played using the keyboard or the button switch inputs.

Adding an electronic output

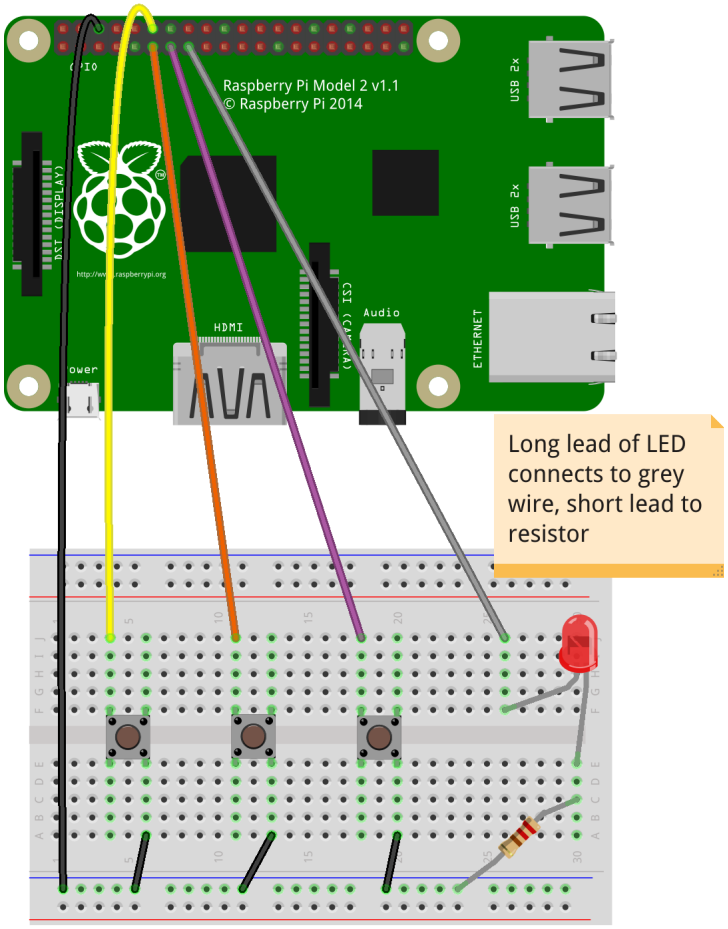
You have already used electronics for the inputs, but now we can add some electronics for the output. In this case we are just going to use a single LED that will light-up each time the asteroid is hit by a missile. The LED is one of the simplest forms of output and can be connected straight to an output pin on the Raspberry Pi GPIO.

The one additional thing that is required is a resistor to limit the amount of current that can flow through the LED which will protect both the LED and the processor inside the Raspberry Pi. In this

example you should use a 220Ω resistor. The resistors are colour coded and for a 220Ω resistor it would normally have 4 bands of colours: red, red, brown and gold.

When connecting the LED then it needs to be connected correctly to work. The LED will normally have one lead that is longer than the other. The longer lead connects to the positive side (anode) which should be connected towards the GPIO pin. The shorter lead connects to the negative side (cathode) which connects towards the ground power supply (in this case via the resistor).

The updated breadboard layout is shown below:



You then need to update the code to turn the LED on for a period of time whenever an asteroid is hit.

To turn on the LED is fairly straight-forward, which can be achieved by adding the following code. First define the new LED near the top of the file along with the other code definitions.

```
PIN_LEFT = 18
PIN_MISSILE = 17
PIN_RIGHT = 27
PIN_LED = 22
button_left = Button(PIN_LEFT)
button_missile = Button(PIN_MISSILE)
button_right = Button(PIN_RIGHT)
```

```
...

missile_status = 0

# The LED output
led = LED(PIN_LED)

# Status 0 = stop, 1 = playing game

...

# When hit reset the asteroid position
def hit_asteroid():
    # Set new position
    asteroid.pos = (random.randint(ASTEROID_SIZE[0]/2, WIDTH-(ASTERO,
ID_SIZE[0]/2)), ASTEROID_SIZE[1])
    led.on()
```

This will turn the LED on, but not off again. You will need to add some code to turn it off again after a short period of time, say ½ second. If you have programmed in python before then you may think of using time.sleep(), but unfortunately that isn't going to work so well in a graphical program where you need the program to continue running. Instead Pygame Zero includes a Clock function which allows you to run another function after the time has expired.

To do this add the following entry to the bottom of the detect_hit function, which will call led.off() ½ second after the LED has been turned on.

```
    led.on()
    clock.schedule_unique(led.off, 0.5)
```

The game is now complete. For now …

Make the game your own

You now have a working game, but that should only be the start. You can now expand the game and improve it however you want. Here are some suggestions, but feel free to customize it however you want.

- Change the speeds of the asteroids / launcher to make it easier or harder
- Have multiple asteroids
- Allow multiple missiles to be fired
- Add good characters that need to be avoided (eg. ships flying across the screen)
- Add more electronics such as 7 segment LED scores, or LCD displays

If you create your own version, please let me know on Twitter @penguintutor and perhaps your ideas will be included as suggestions on the on the web page: http://www.penguintutor.com/space-asteroids

Source Code

The complete code is shown below:

```
from gpiozero import Button, LED
import random

WIDTH = 600
HEIGHT = 500

PIN_LEFT = 18
PIN_MISSILE = 17
PIN_RIGHT = 27
PIN_LED = 22
button_left = Button(PIN_LEFT)
button_missile = Button(PIN_MISSILE)
button_right = Button(PIN_RIGHT)

# The Launcher
launcher = Actor('launcher')
launcher.pos = int(WIDTH/2),HEIGHT-50
launcher_speed = 2

# The Asteroid
ASTEROID_SIZE = 50,46
asteroid = Actor('asteroid')
asteroid.pos = (random.randint(ASTEROID_SIZE[0]/2, WIDTH-
(ASTEROID_SIZE[0]/2)), ASTEROID_SIZE[1])
asteroid_speed = 1

# The missile
MISSILE_SIZE = 20,43
missile = Actor('missile')
# initially set missile position in top corner
missile.pos = 0,0
missile_speed = 10
# Status 0 = not fired, 1 = fired
missile_status = 0

# The LED output
led = LED(PIN_LED)

# Status 0 = stop, 1 = playing game
game_status = 0
game_score = 0

def draw():
    if game_status == 0:
        screen.draw.text("Press ENTER to start", (100, 300),
color="white", fontsize=32)
    if game_status == 1:
```

```
        screen.clear()
        # display score
        screen.draw.text("Score: " + str(game_score), (WIDTH-100, 50),
color="blue", fontsize=24)
        launcher.draw()
        # Each draw loop move asteroid down 1 position
        move_asteroid()
        asteroid.draw()
        # If missile launched move missile up and show
        if missile_status > 0 :
            move_missile()
            missile.draw()
        # Detect if missile / asteroid has hit etc
        detect_hits()

def update():
    global game_status, game_score, missile_status, asteroid_speed
    if game_status == 0:
        if keyboard.RETURN:
            game_status = 1
            asteroid.pos = (random.randint(ASTEROID_SIZE[0]/2, WIDTH-
(ASTEROID_SIZE[0]/2)), ASTEROID_SIZE[1])
            game_score = 0
            asteroid_speed = 1
    elif game_status == 1:
        # Keyboard movement
        if keyboard.right or button_right.is_pressed :
            move_launcher(launcher_speed)
        if keyboard.left or button_left.is_pressed :
            move_launcher(-1 * launcher_speed)
        # If missile not fired then it can be fired
        if missile_status == 0:
            if keyboard.space or button_missile.is_pressed :
                # set position to location of launcher
                missile.pos = (launcher.x, HEIGHT - 50)
                # set it fired
                missile_status = 1


# Move launcher allowing, prevent overrunning
def move_launcher(distance):
    newpos = launcher.x + distance
    if (newpos < 50):
        newpos = 50
    elif (newpos > WIDTH-50):
        newpos = WIDTH-50
    launcher.x = newpos


# Move asteroid position and check for hits
def move_asteroid():
    global game_status
    # Move position down
    asteroid.pos = (asteroid.x, asteroid.y+asteroid_speed)
```

```
# If missile launched then move it up appropriate distance
def move_missile():
    global missile_status
    # Only move / draw if missile is fired
    if missile_status > 0:
        missile.pos = (missile.x, missile.y - missile_speed)
        # Near top and not collided so stop missile
        if missile.y < 10:
            missile_status = 0

# Check for hits between asteroid and ground / missile
def detect_hits():
    global game_status, game_score, missile_status, asteroid_speed
    # Check if we have hit the ground (game over)
    if asteroid.y >= HEIGHT-(ASTEROID_SIZE[1]/2):
        screen.draw.text("Game Over", (100, 100), color="red",
fontsize=32)
        # Stop the game
        game_status = 0
    # check if missile has hit asteroid
    if missile_status == 1 and asteroid.collidepoint(missile.pos):
        # When hit asteroid add point and start new
        hit_asteroid()
        missile_status = 0
        # increase asteroid_speed
        asteroid_speed += 1
        game_score += 1


# When hit reset the asteroid position
def hit_asteroid():
    # Set new position
    asteroid.pos = (random.randint(ASTEROID_SIZE[0]/2, WIDTH-
(ASTEROID_SIZE[0]/2)), ASTEROID_SIZE[1])
    led.on()
    clock.schedule_unique(led.off, 0.5)
```