Space Asteroids – Scratch

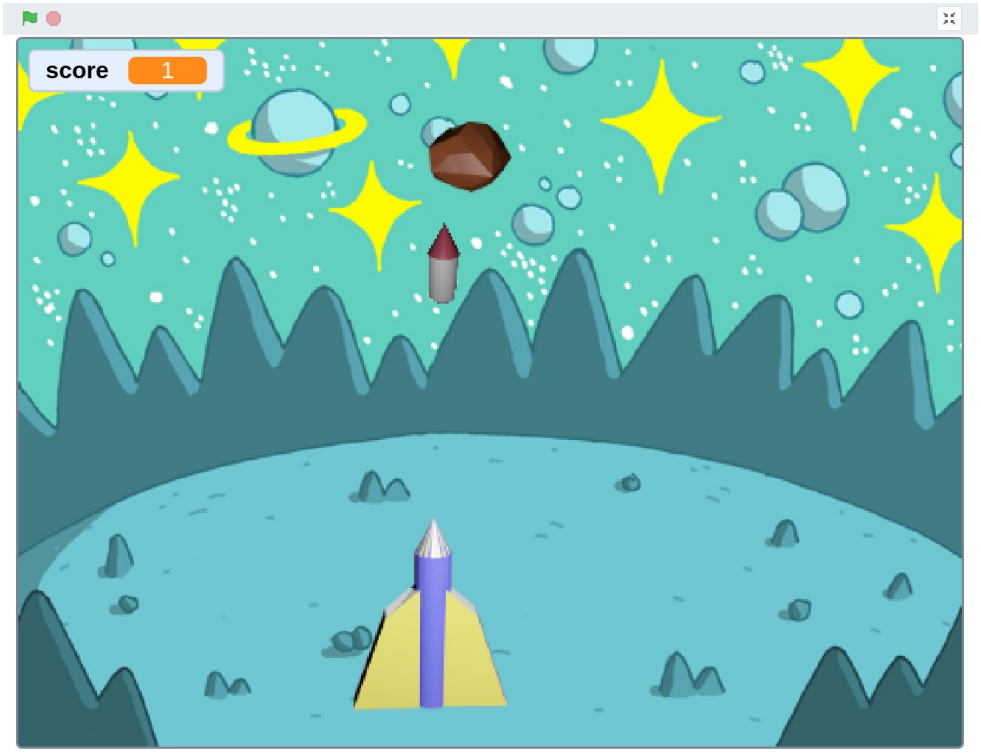
Physical computing and game programming with the Raspberry Pi

Updated for Scratch 3

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 which uses the optional sprites:



You will need:

Raspberry Pi 4 or later (min 2GB RAM)
Breadboard
3 x Push-to-make PCB switches
LED
220Ω resistor
Male-to-female and male-to-male jumper leads

Getting started

This version is based around Scratch 3. If not already installed, it can be added from the recommended software or using:

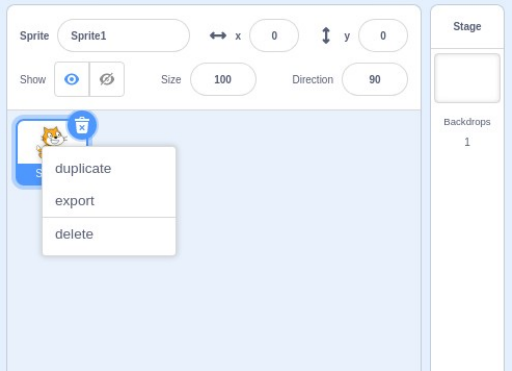
```
sudo apt install scratch3
```

It can then be launched from the programming menu on the Raspberry Pi.

Tip: You should give your project a name and save it regularly so that you don't lose any progress if there is a problem.

Remove the cat sprite

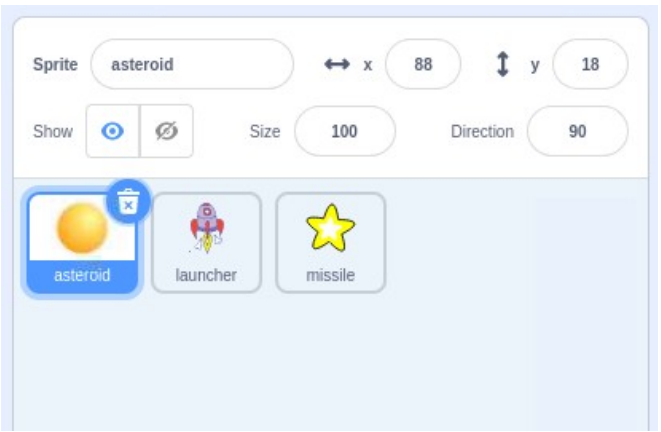
When starting Scratch there is a cat shown on the stage. We don't need a cat for this game so it can be deleted from the sprites area by clicking on the bin icon or right-clicking and choosing delete.



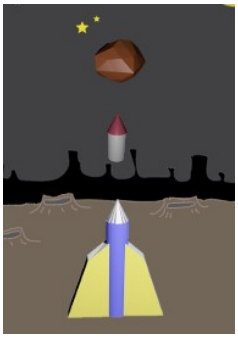
Add new sprites

Add three new sprites using the new sprite button:

Choose the following (or any other suitable images): Ball (this will be called asteroid), Rocket Ship (this will be called launcher) and Star (which will be called missile). Change their names using the "Sprite" field.



Alternatively there are Sprite images which can be downloaded from:

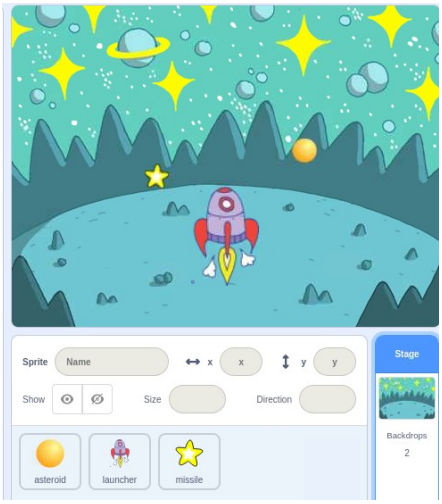


www.penguintutor.com/space-asteroids.

Change the size of each of the sprites by setting their size to 60.

Set the background

You can now choose a suitable background using the stage, new backdrop button. The space background is a good choice.



Controlling the launcher with the keyboard

The launcher needs to move from side-to-side. First you will set this up using the keyboard and then add the electronics to create your own controller. This needs code to be added to each of the sprites. To make it clear which sprite the code needs to be added to the sprite name is highlighted before each code example.

launcher

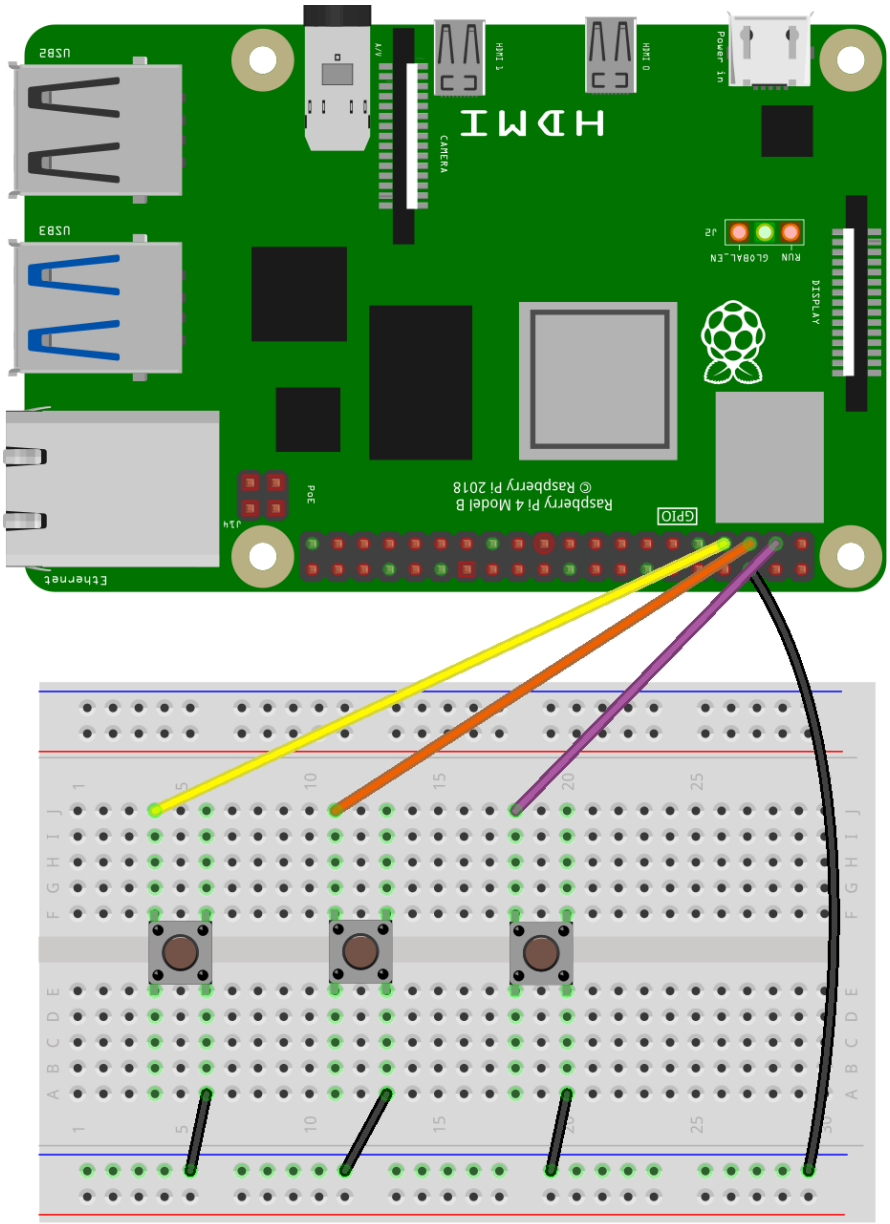
Click on the launcher sprite. Create the code shown below inside the large code area in the middle of the screen. Note that the different colour blocks corresponds with the different block categories.

This code will run whenever the green flag (start program) is clicked. It sets the position of the launcher to the bottom middle and then loops forever moving left if the left-arrow is pressed and right if the right-arrow is pressed.
Scratch automatically stops when it reaches the side of the screen, so there is no need to add any code to check for that.

You should be able to test this by clicking on the green arrow above the stage and using the left and right arrow keys on your keyboard.

Adding the Electronic Switches

You can now add the three buttons (switches) to the breadboard and connect them to the Raspberry Pi GPIO pins. There are two buttons to move the launcher left and right, which you will code up now and one to launch missiles which comes later.



Check you wire these correctly to the Raspberry Pi. The colour of the wires does not matter as long as they go to the correct position; for example the ground wire (shown as black) goes to the top row 3rd pin from the left and the wire to the first switch (shown as yellow) goes to the bottom row 2nd pin from the left.

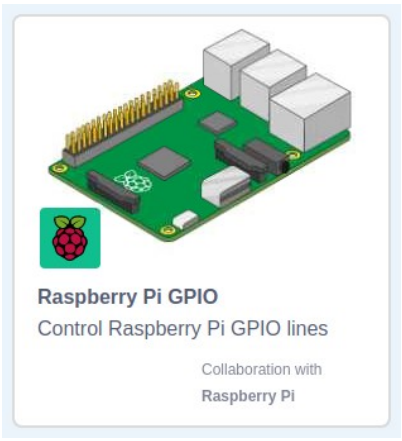
Note: The Raspberry Pi is shown intentionally upside down, with the GPIO ports along the bottom. This is so that the video cable does not get in the way of your wiring.

The switches connect the GPIO input pin down to ground (0v). The way that this works is due to an internal pull-up resistor inside the Raspberry Pi. When the switch is not pressed then the pull-up resistor (which is within the processor so cannot be seen) takes the input to a high voltage. When the button is pressed then it connects the input pin to ground which is at zero volts. As the resistance through the switch is much less than the pull-up resistor this sets the GPIO input pin to be a low value.

The Raspberry Pi records these as two values:
Switch not pressed = High
Switch pressed = Low

Once the circuit is wired together you can update the code to look for the High and Low values. First you need to enable the GPIO extension by clicking on Add Extensions in the bottom left of the screen.

Choose Raspberry Pi GPIO and click OK (this only needs to be done once and then this is permanently enabled for this program).



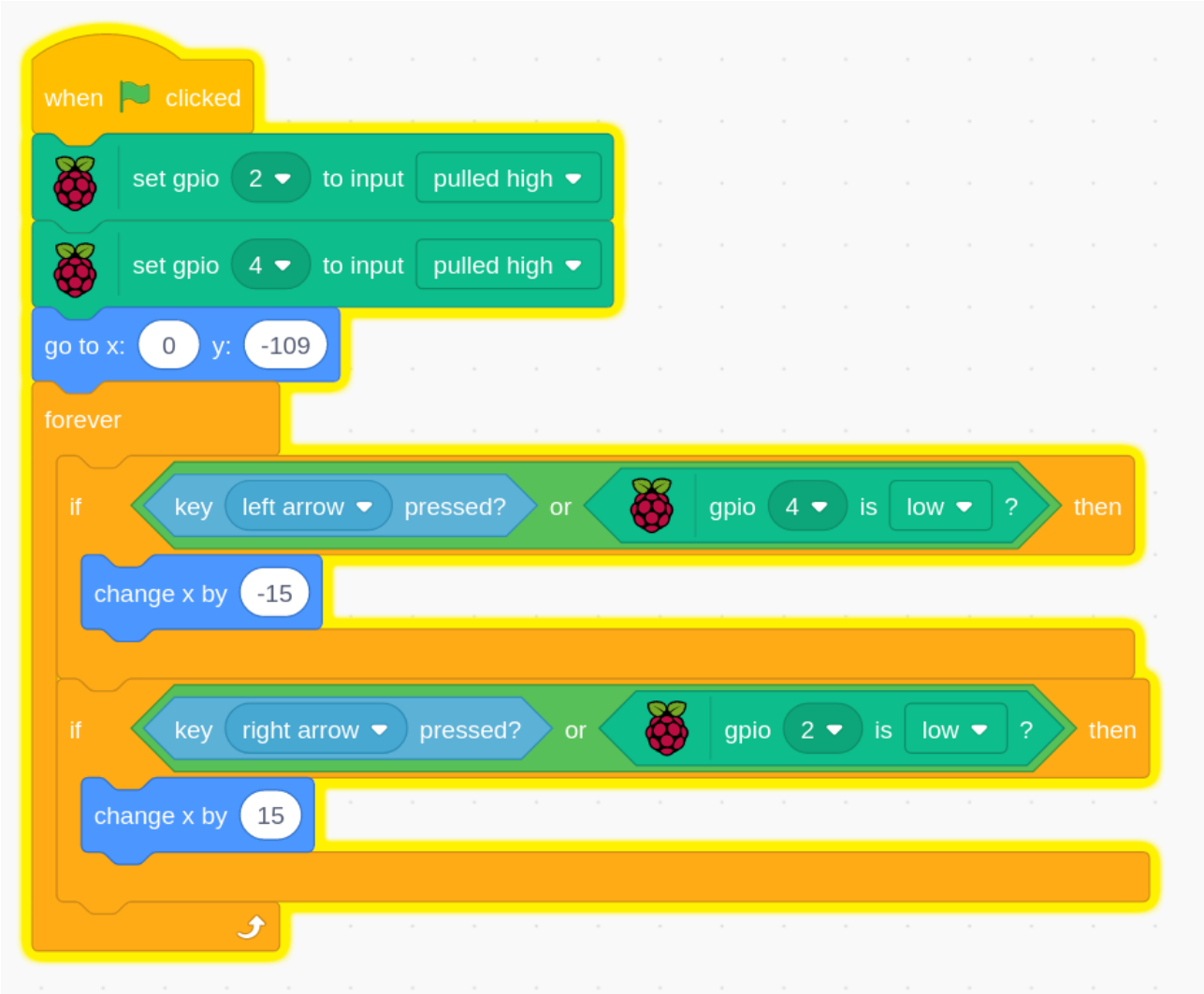
This will add Raspberry Pi GPIO blocks below the “My Blocks” category. These include the ability read the state of the GPIO ports (used with switches) or set the output of GPIO ports (used with the LED later).

The GPIO pins we will be using are 2 for moving to the left and 4 for moving to the right. Pin 3 will be used for firing the missile.

> The pins on the GPIO are labelled in terms of how they connect to the processor rather than the numbering on the Raspberry Pi.
> See www.penguintutor.com/raspberrypi-gpio for more details.

You can replace the existing keyboard sensing blocks with the GPIO block, or add an “OR” operator as I have done in the example below.

launcher



Remember the GPIO is set to high normally, and will go low when the button is pressed.

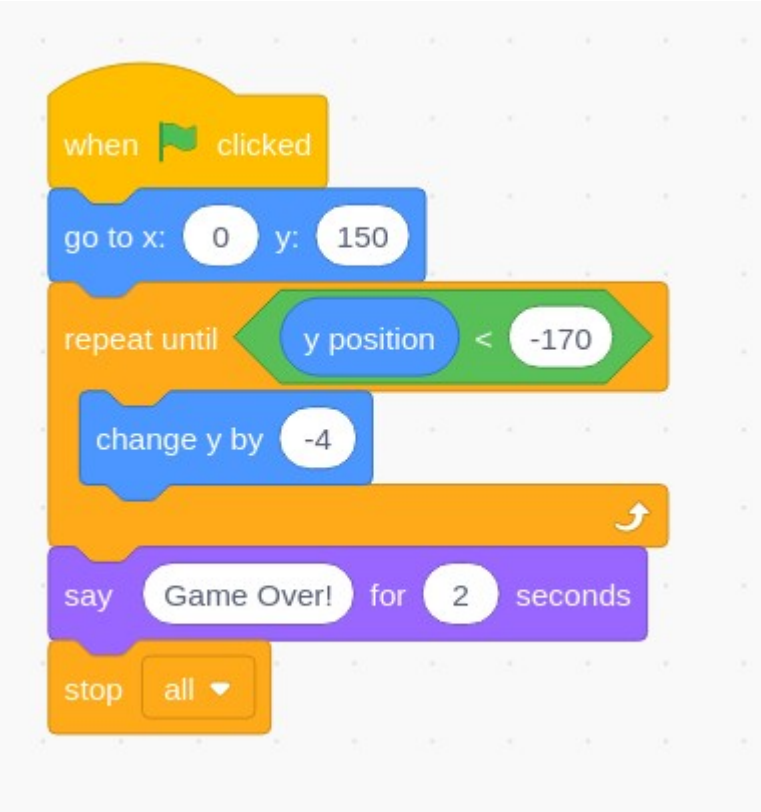
If you click on the green flag to start the program you should now be able to move the launcher left and right using the keyboard arrows or the buttons you’ve wired up to the Raspberry Pi.

Asteroids and Gravity

At the moment the asteroid is just hanging in the air, but instead you need to add code to have gravity pull it down to earth.

asteroid

This can be added by clicking onto the asteroid and then going to the asteroid's code area. Then add the following code:



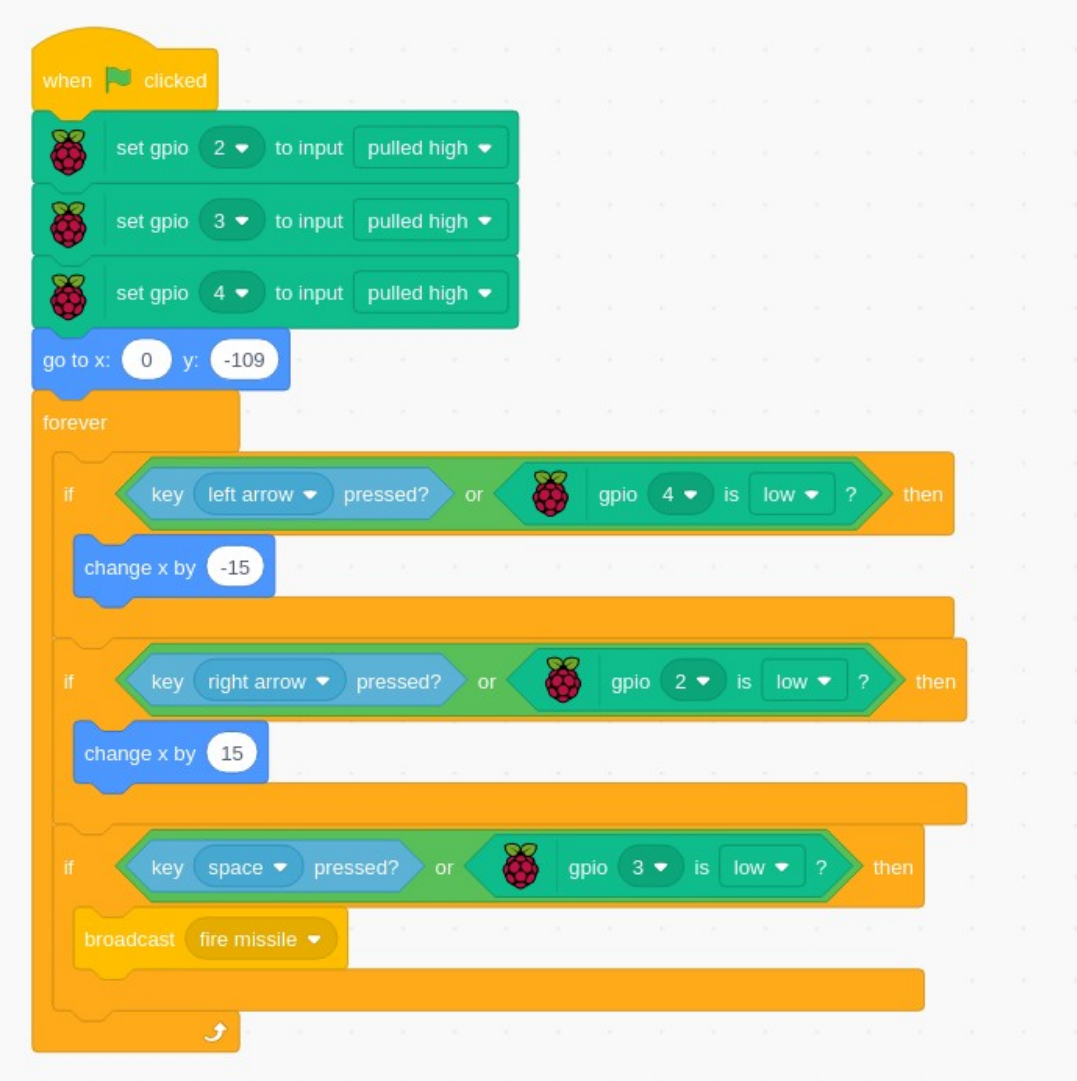
This places the asteroid in the top middle position. It then has a loop which will keep running until the asteroid reaches the bottom of the screen. During the loop the asteroid will keep moving down by 4 steps at a time.

When the asteroid reaches the bottom then it will say game over and then stop the game.

Controlling the missile

You now need some way of firing a missile at the asteroid. This needs changes to both the launcher scripts, so that it can fire the missile and to the missile itself. Start by clicking on the launcher and editing the previous script to add the parts related to the missile as below:

launcher

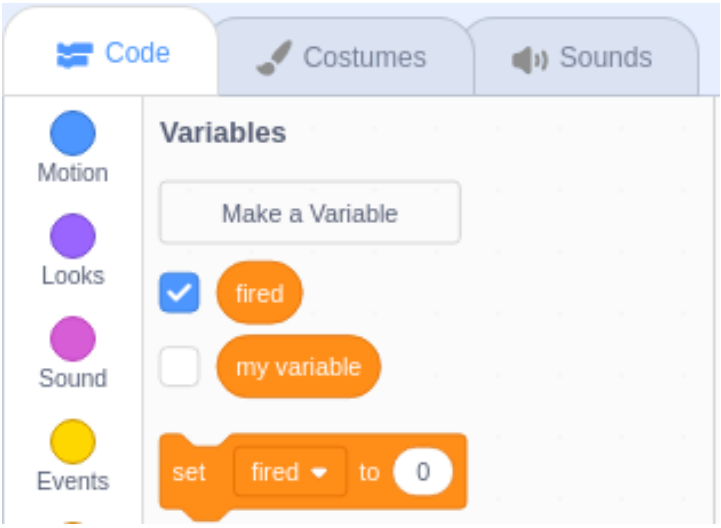


Most of this is similar to the code used to detect the left and right buttons. The new thing is the “broadcast” block. The broadcast is used in Scratch to send a message between scripts that are running on different sprites, in this case from the launcher to the missile. When you add the broadcast block then you need to first choose “new message ...” and type in the message “fire missile”.

One extra thing we are going to need is a variable which will keep track of whether the missile has been fired or not. This will prevent us from launching a missile until the previous one has gone off the screen or hit an asteroid. To create the variable go to the missile sprite and in the script tab choose data and add Make a Variable.

missile

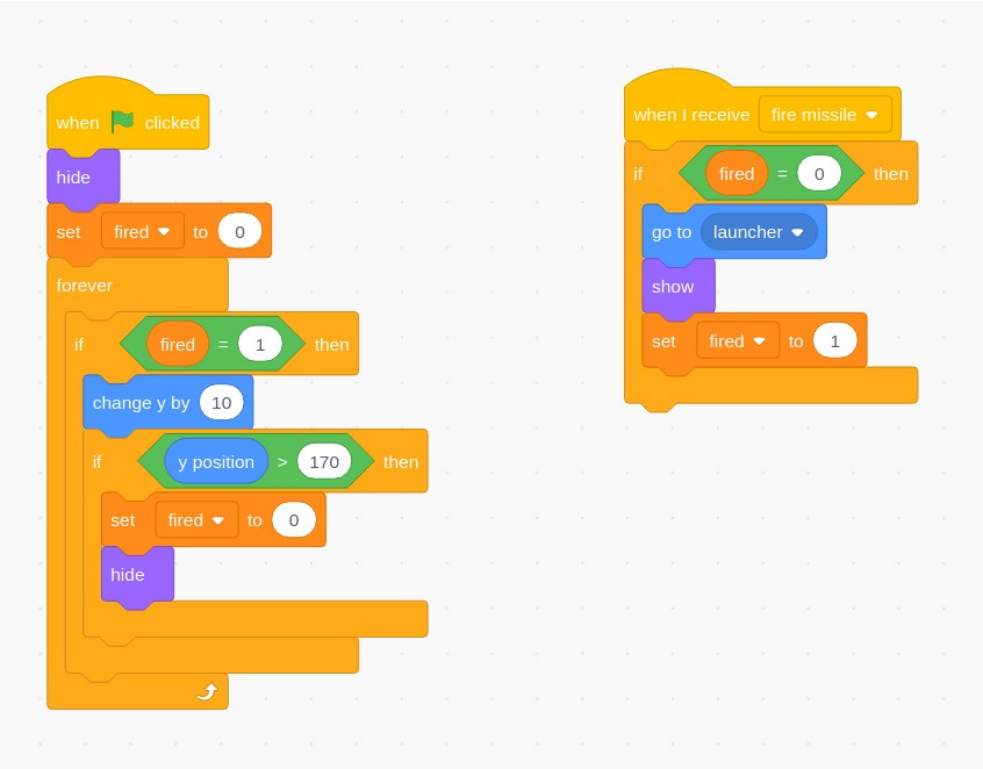
Enter the variable name “fired” and select for this sprite only and then click OK.



By default the variable will be shown on the stage, but as this is an internal variable we don't want the user to see it, so untick the box next to the new variable.

Now add the following code into the missile's code area.

missile



The 

block of code first hides the missile (we don't want to see it until it's been launched) and sets the fired status to 0 (not fired). It then runs a loop which first checks to see if the missile has been fired, in which case move it up the screen. It then checks to see if the new position is at the top of the screen, in which case it sets the fired back to 0 and hides the missile.

The block 

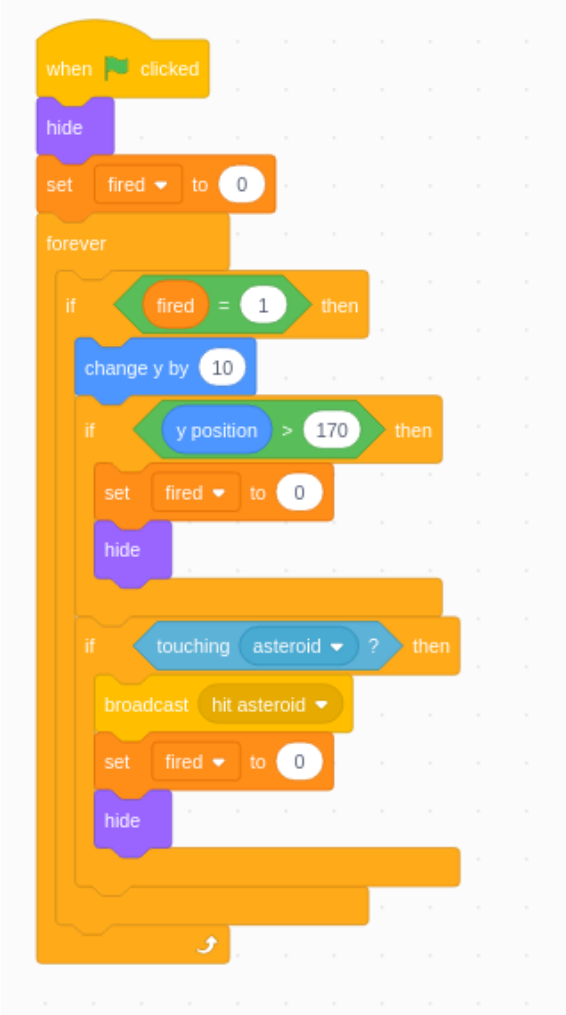
runs whenever it receives the broadcast message from the launcher. Assuming the missile has not recently been fired, it then sets the position to be the top of the launcher, makes it visible and then sets the fired status to 1 (so now it will be moved up by the previous block of code).

Detecting a hit

The next thing is to detect when the missile hits the asteroid. To implement this the code needs to updated on the missile and the asteroid sprites so that it can trigger a new asteroid once the previous one is destroyed. You will once again need to add a new broadcast message to communicate between the two sprites.

Start by updating the missile's code to detect if the missile touches the asteroid.

missile

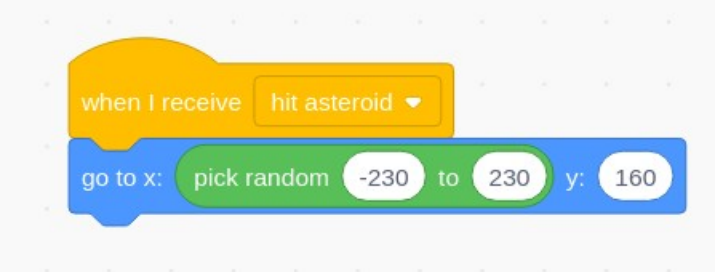


You just need to add the code from “if touching Asteroid” downwards.

This code detects if the missile hits the asteroid and if it does then sends the broadcast message “hit asteroid” and sets the fired status to 0 and hides the missile.

Next on the asteroid add the following code block in addition to the one that is already there.

asteroid

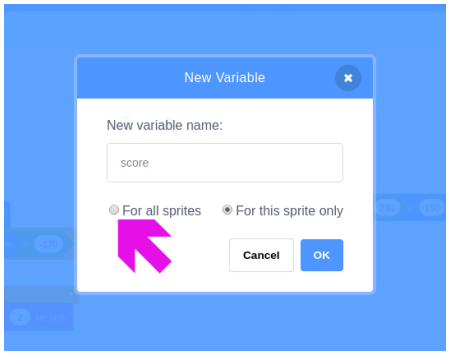


At the moment this just moves the asteroid to a random position near the top of the stage. This means that the next asteroid could appear at any position from the left to the right of the stage, or anywhere in between.

Keeping score

You now have a playable game, but to be able to compare how well you did with your friends then it's a good idea to add a score variable to track how many asteroids you manage to hit.

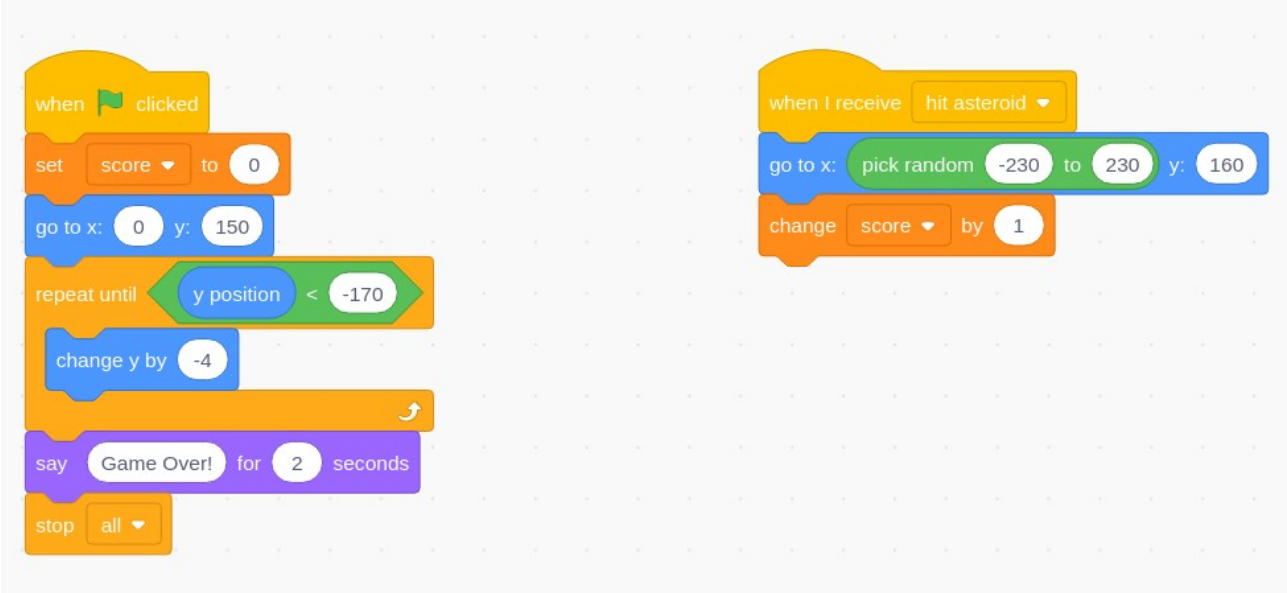
You will need to create a new variable called score. This can be created “for all sprites” so it can be created under any of the sprites.



Next update the asteroid's scripts to add the two new score related blocks shown below:

The first is to set the score to zero when starting the game and then the other increases the score every time that the missile hits an asteroid.

asteroid

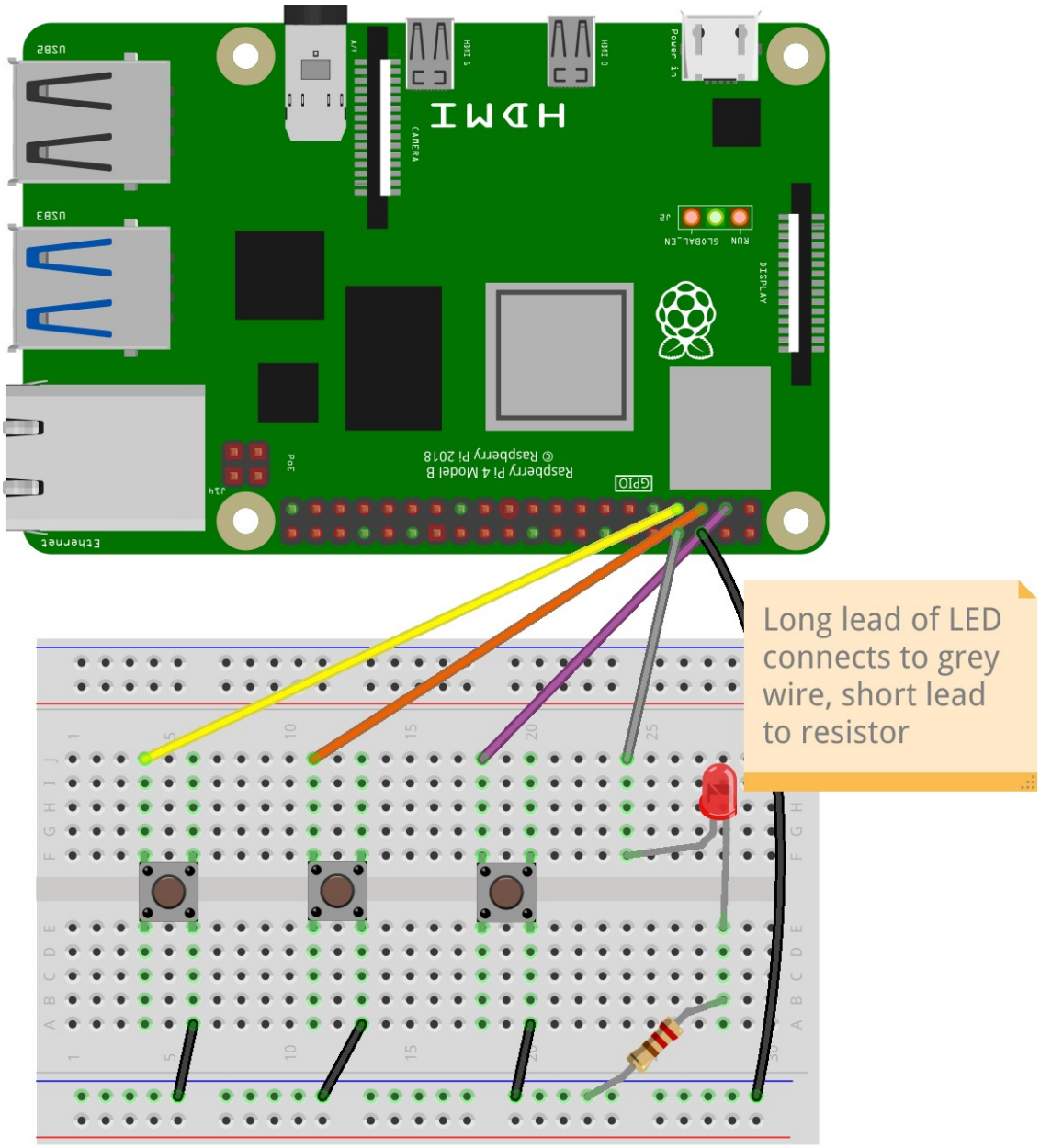


You should now have a working game that you can play.

Add an LED output

You have used electronic inputs in the form of some push button switches, but now it's time to add an output. The most simple form of output is an LED which lights up when a GPIO pin is set high. As well as the LED you will also need to add a 220Ω resistor. The resistor limits the amount of current to protect the LED as well as the Raspberry Pi. The resistors are colour coded and for a 220Ω resistor it would normally have 4 bands of colours: red, red, brown and gold.

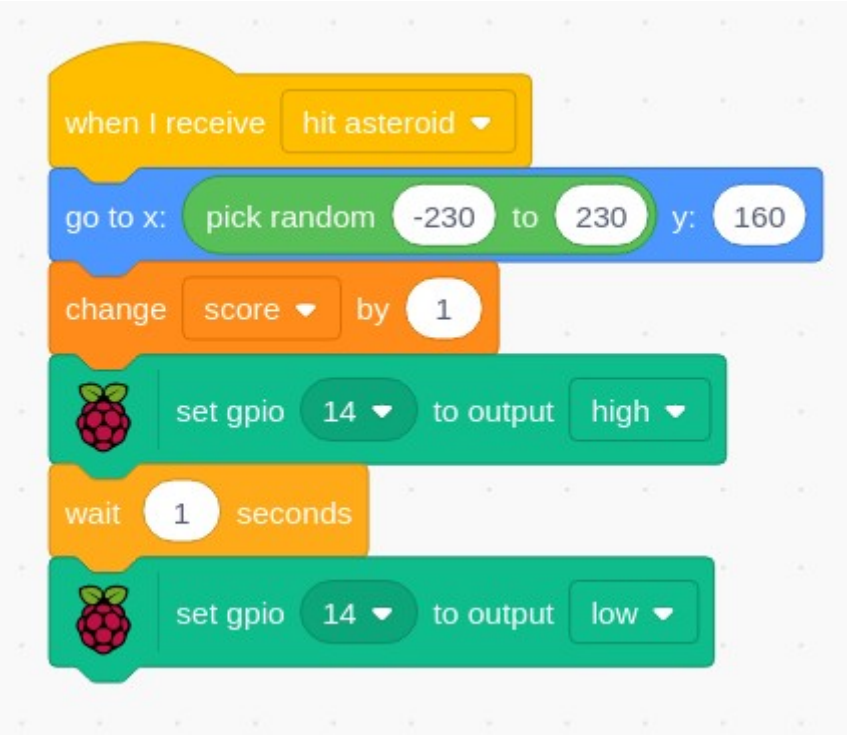
The LED and resistor should be added to the circuit as shown below:



The resistor can be connected either way around, but the LED needs to be connected so that the longer lead is connected to the breadboard where the grey wire is and the shorter lead to the resistor.

Turning the LED on and off is as simple as turning the GPIO port high and low. You can do this by updating the existing script on the Asteroid.

asteroid



Making the game your own

The game is now complete. If you fancy a challenge then you could update the code to customize it yourself. You could create your own custom sprites, or have the asteroids change speed to make it harder (or easier).

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