Puzzle room

Create a spaceship puzzle room with a character that solves puzzles.

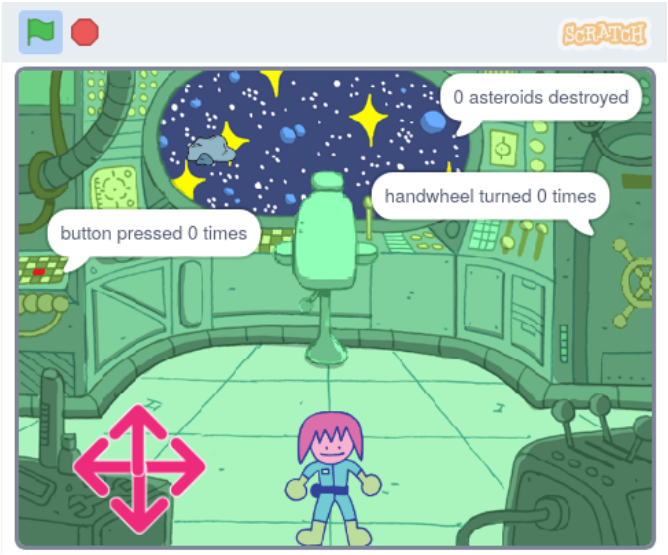
Step 1 Introduction

In this project, you will create a spaceship puzzle room with a character that solves puzzles.

> **Puzzle rooms**, or **escape rooms**, are rooms containing puzzles that you must solve to get out of the room, or to get into the next room. They can be online or real rooms, but the idea is usually the same: complete the mission as quickly as you can!

You will:

- Use a `repeat until` block to control when an action stops
- Use multiple `join` blocks to output the value of a `variable`
- Combine `and`, `or`, and `not` conditions to solve problems



Step 2 Who's in the puzzle room?

In this step, you will add a character to a puzzle room, and create controls to move them around.

Open **the Scratch starter project** (https://scratch.mit.edu/projects/531567946/editor/). Scratch will open in another browser tab.

If you are working offline, you can download the starter project at **rpf.io/p/en/puzzle-room** (https://rpf.io/p/en/puzzle-room).

Working offline

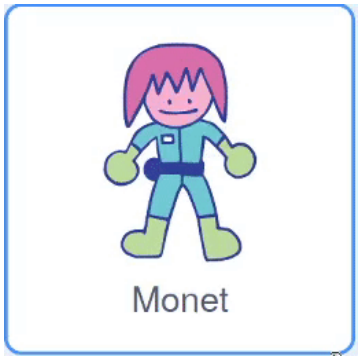
For information about how to set up Scratch for offline use, visit **our 'Getting started with Scratch' guide** (https://projects.raspberrypi.org/en/projects/getting-started-scratch/1).

You should see a scene from inside a spaceship. Several sprites have been made for you, and their positions have been set.

Choose: Who's in the spaceship? It could be a solo mission from Earth, it could be an alien spaceship, or it could even exist in a future where cats rule the world.

You need one character to interact with the puzzles you are making.

Add a new sprite to your project. In this example, you will see the character **Monet**.

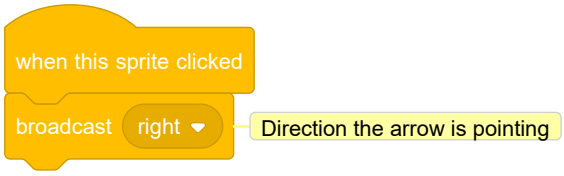


If your character is too large or too small for the scene, you should change their size. You can also pick a starting position for the character.

Add code to set up your character's size and starting position.

You will need on-screen controls to enable you to move your character around.

Select one of the **arrow** sprites. When the sprite is clicked, it should broadcast its direction so that it can make the character move in that direction.



Tip: If the **arrow** sprites are too difficult to click on when using a mobile or tablet, then you can change their costumes. Each **arrow** sprite also has a large purple circle that can be used instead.

Now add more direction controls to move your main character.

Duplicate the **arrow** sprite three times. Then, for each sprite, change the costume so it points in a different direction.

Change each sprite's name to the direction it is pointing and change the `broadcast` to the direction it is pointing.

Arrange all the **arrow** sprites in the corner of the screen.

Your main character should move when the arrows are pressed.

Code your main character sprite to move when it receives broadcasts to go `left, right, up, and down`.

Test: Click the green flag and then click on the arrows to move your character around.

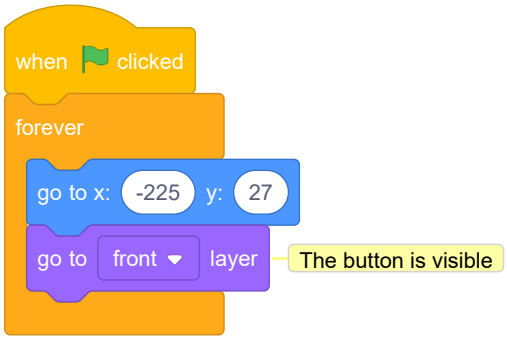
Save your project

Step 3 The button puzzle

In this step, you will add your first puzzle, which will be to push the button a certain number of times.

When the game starts, the button needs to stay in the same place, and always be visible on the front layer.

Add the following blocks to the **button** sprite.

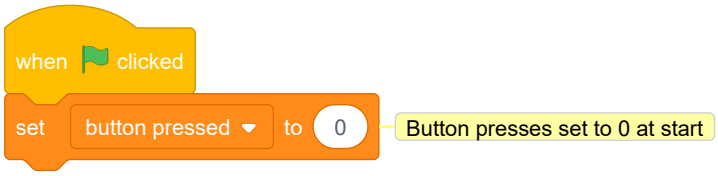


The button will need to be pushed a number of times for the puzzle to be completed. For this, you will need a `variable` to store the number of pushes.

Create a new `variable` and call it `button pressed`.

At the start of the game, `button pressed` should be `0`.

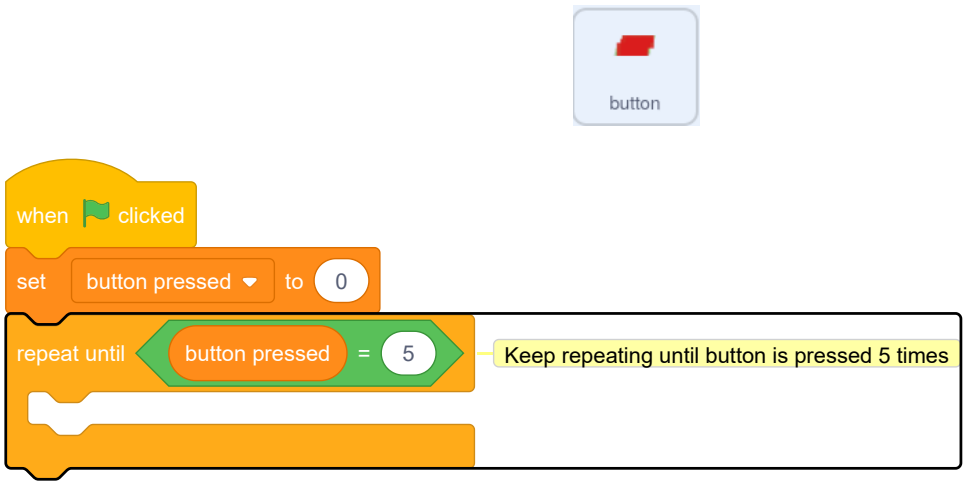
Add the following blocks to the **button** sprite.



A `repeat until` block is a loop that keeps repeating until a certain condition is met.

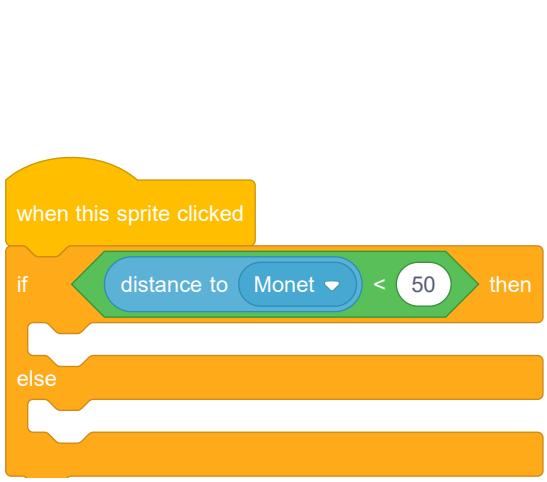
Choose: How many times will the button need to be pressed to solve the puzzle? In this example, it will need to be pressed `5` times, but you can choose a different number.

Add a `repeat until` loop, and set its condition to be when `button pressed` is `equal` to `5`.



Now, the player needs to be able to push the button. They should only be able to press it when the character is close to the button though!

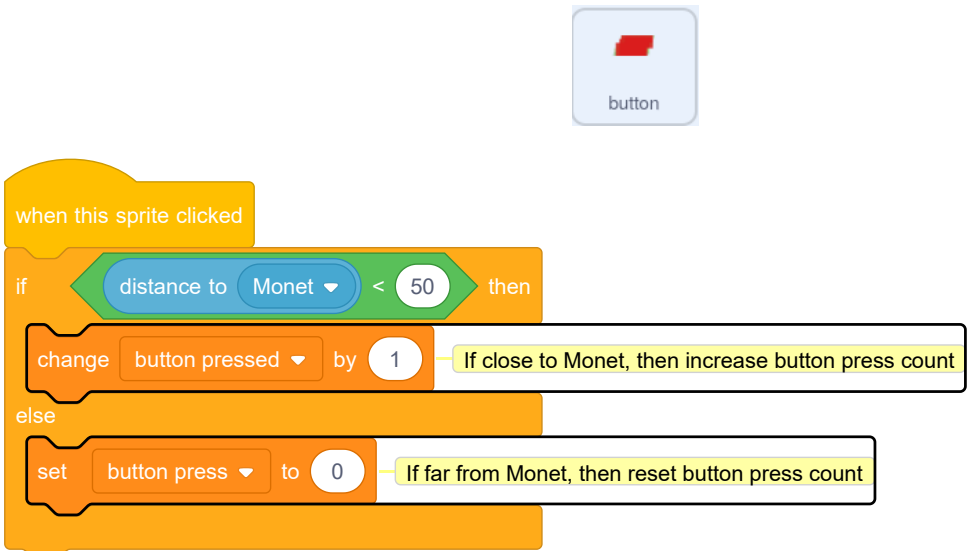
Add blocks to detect if the character is close to the button when the **button** sprite is clicked.



If the character is close, and the button is pressed, then the `button pressed` variable can be increased. If the character is not close, the puzzle should reset; the player needs to push the button five times in a row, before trying any other puzzles.

Tip: In Scratch, the distance between any two sprites is calculated from the centres of the sprites. This means that large sprites can look as if they are touching, but their centres may still be far apart.

Add code to change the value of the variable `button pressed`.

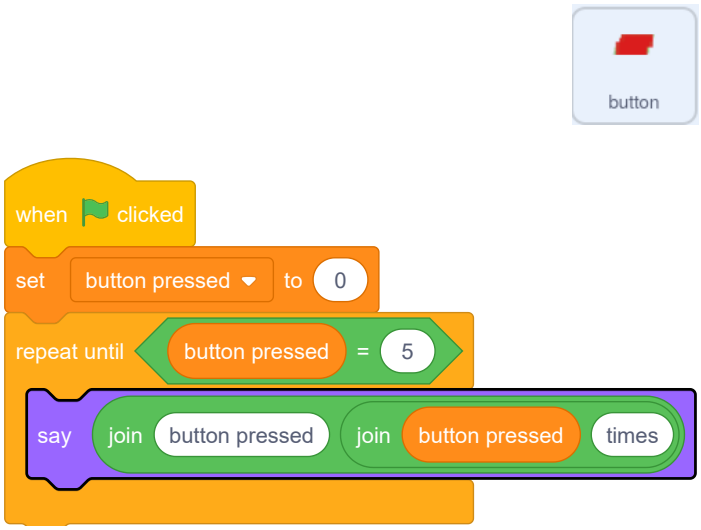


Test: Run your project and move the character close to the button. As you click on the button, the `button pressed` variable should increase. You can adjust the value of `distance to Monet` up or down, until you find a number that makes sense to you.

You can use the `join` block to `say` to the player how many times the button has been pressed.

Place a `join` block inside another one. Then add in the text you want, and the `button pressed` variable, all inside a `say` block.

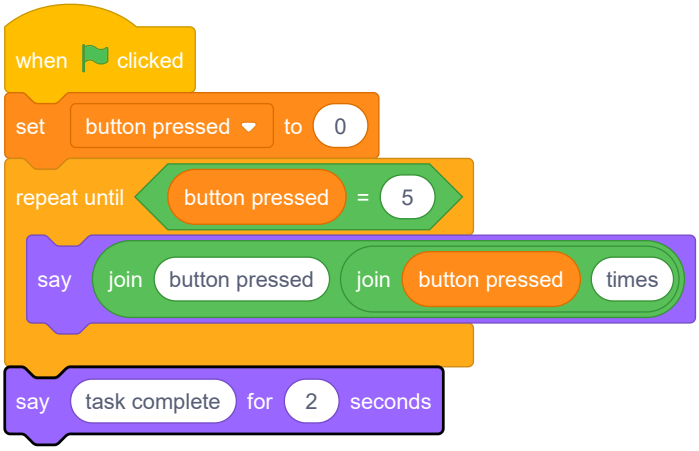
For example:



Tip: Make sure you add spaces in the text in your `join` block.

The loop will end when the button has been pressed `5` times, then the last block in the script will be run. This can tell the player that the task is complete.

Use a say block to tell the player the task has been completed.



Test: Run your project and move the character close to the button. When you click on the button five times, the task should be complete.

Save your project

Step 4 The handwheel puzzle

In this step, you will create a puzzle where a handwheel needs to be turned.

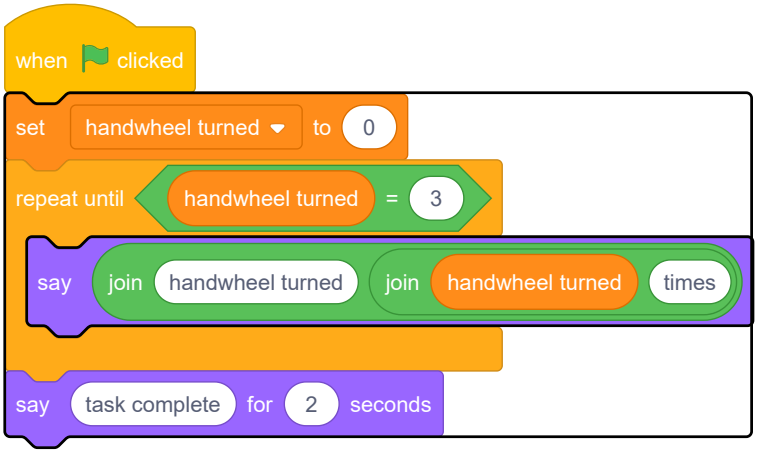
The scripts for this puzzle are quite similar to the button puzzle, so you can copy those scripts over and then edit them.

Drag the two scripts you created for the **button** sprite on to the **handwheel** sprite, to copy them to that sprite.

The `when flag clicked` script is the first one that needs to be changed.

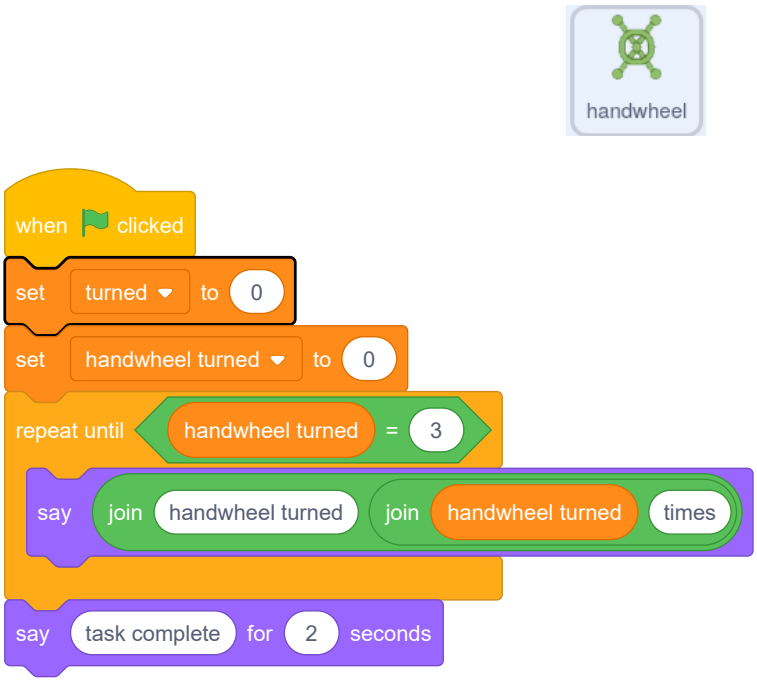
Create a new variable called `handwheel turned`, and use that variable instead of the `button pressed` variable.

Choose: Change the completion number to whatever you would like it to be. We chose `3` in the example.



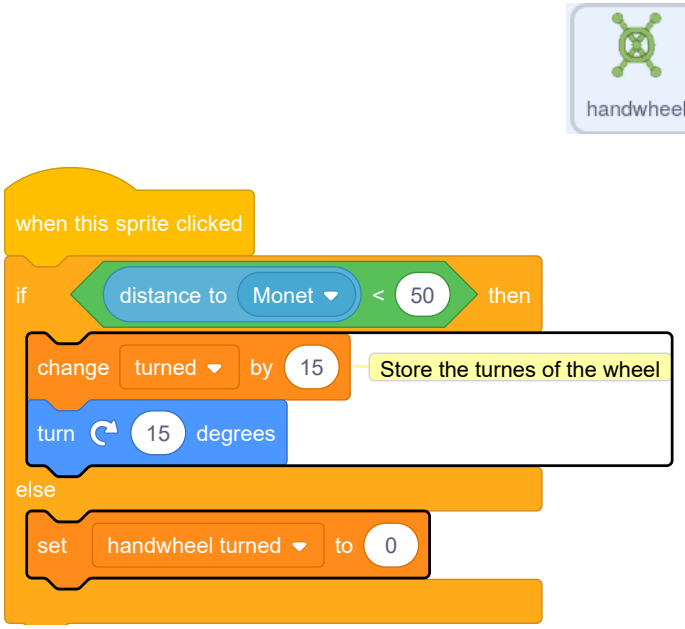
Like a real handwheel, the **handwheel** sprite will only be able to turn a small number of degrees at a time, so the angle it has been turned will need to be stored.

Create a new variable called `turned` and set it to `0` when the game starts.



Now you can edit the `when this sprite clicked` script, so that when the **handwheel** sprite is clicked repeatedly it turns a small amount each time until it completes a full revolution. When it has completed the right number of full turns (`3` times in the example), the puzzle will be solved.

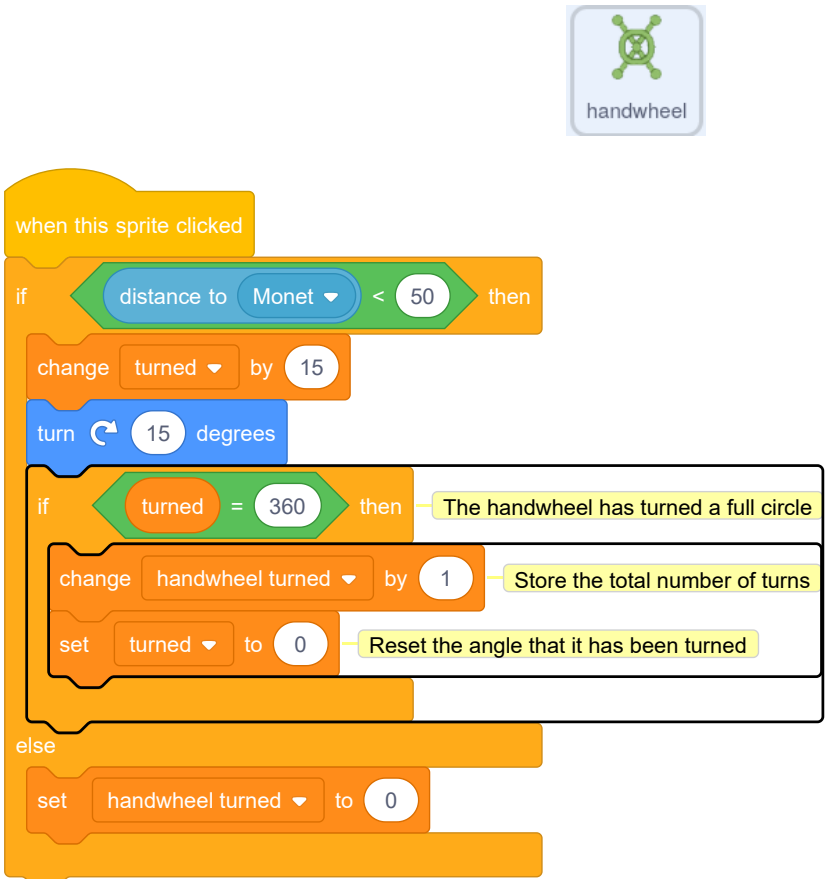
Add blocks so that each time the **handwheel** sprite is clicked, it turns `15` degrees and the `turned` variable increases by `15`.



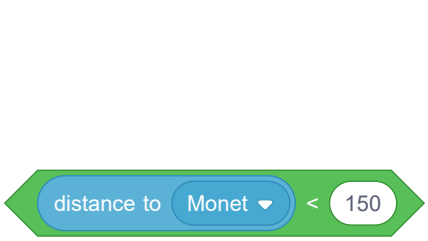
Test: Move the **Monet** sprite (or your character sprite) close to the **handwheel** and then click on the **handwheel** sprite. It helps to be in fullscreen mode, so that you can't drag the **handwheel** sprite around.

When the `turned` variable reaches `360`, then the handwheel has been turned once; this can now be stored in the `handwheel turned` variable.

Use a **nested** `if` to change the `handwheel turned` and reset the `turned` variables. A **nested** `if` is when one `if` is placed inside another.



Test: Move your character sprite close to the handwheel, and then click on it. You might have to adjust the distance that the character needs to be from the handwheel.



Tip: You can click and drag your **Monet** (or character) sprite around, to bring it closer to the handwheel, for instance. This will save you time, as you won't keep on having to use the controls.

Save your project

Step 5 The asteroids puzzle

In this step, you will create the most challenging puzzle. You will create a puzzle to destroy dangerous asteroids.

You will need a crosshair that you can use to target the asteroids.

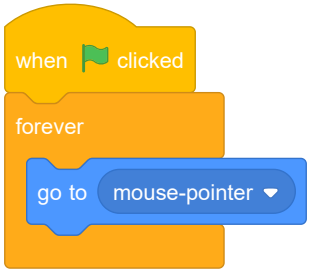
Paint a new sprite called **crosshair**. An example is shown below, using a circle and two lines. Make the circle solid initially and then adjust its **fill** to transparent, once you have it sized and positioned.

Resize your **crosshair** sprite if you need to.

Tip: You can zoom in on the paint editor, using the **+** and **-** symbols, to make positioning easier, especially if you are on a mobile or tablet device.

The crosshair will follow the mouse, but it should only be visible through the window into space.

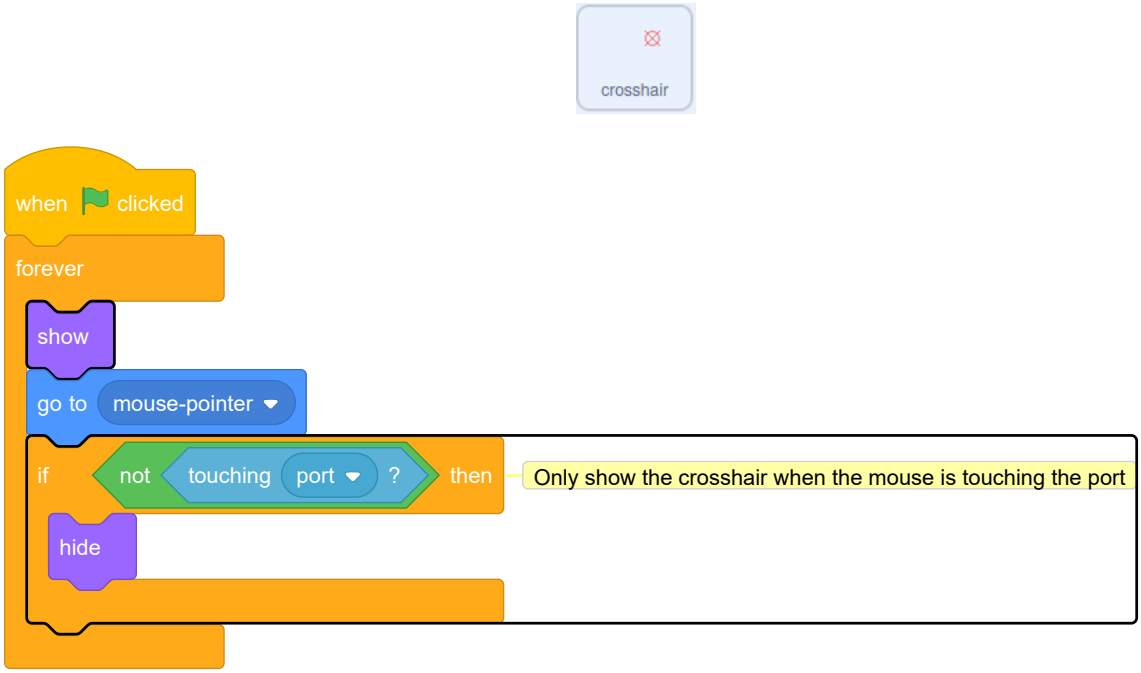
Use the following blocks so that the **crosshair** follows the mouse-pointer.



Test: Click the green flag and make sure that the **crosshair** follows the mouse-pointer.

You can use an block to test if the **crosshair** is touching the **port** sprite, so that it is hidden when it is **not** touching it.

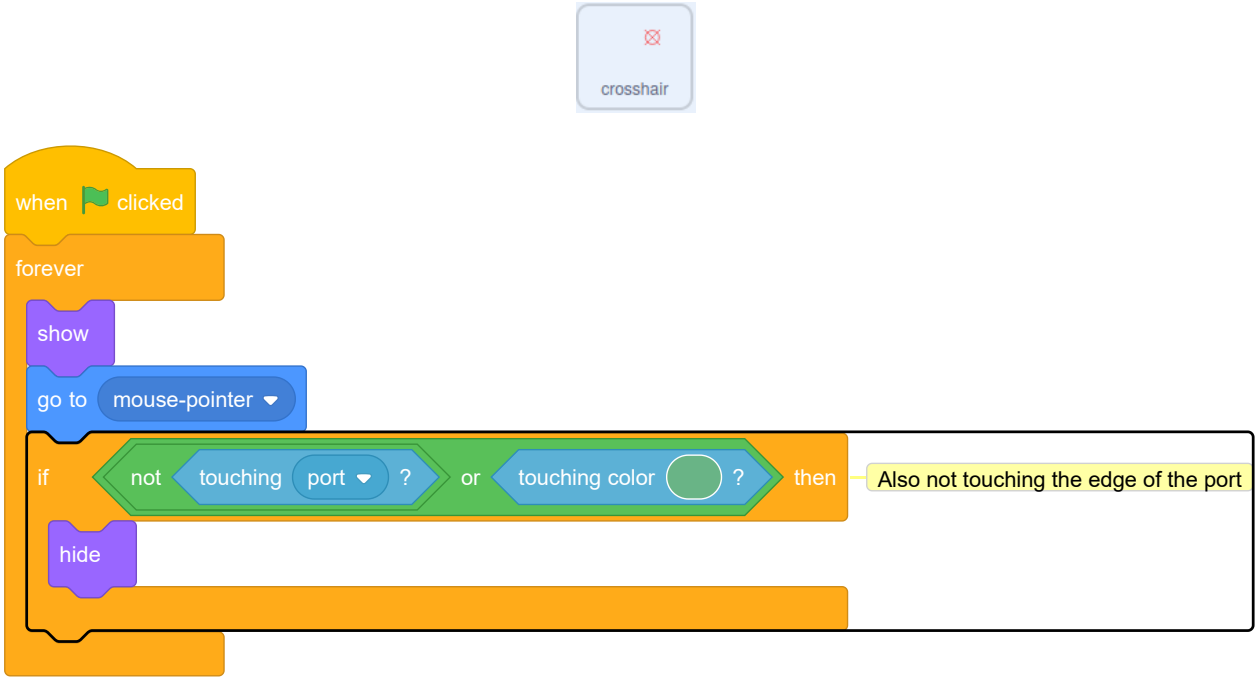
Add a test, to make sure that the **crosshair** is touching the **port**.



Test: Click the green flag and make sure that the **crosshair** hides when it is not touching the **port**.

You might notice that the crosshair appears at the very edge of the **port** and so appears to be inside the spaceship. This can be fixed by checking it's not touching a colour of the backdrop.

Add an or block to the if block. The second condition is if the **crosshair** is touching the green colour that surrounds the **port**.



Tip: As the crosshair follows the mouse-pointer, make sure you stop your project before using the colour picker.

Now it's time to create the asteroids.

Find a **Rocks** sprite and add it to your project. Resize the sprite so that it is not too big.

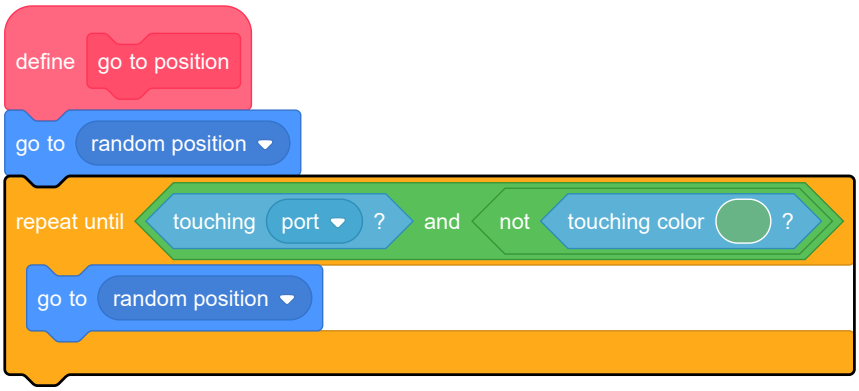
You may have seen how `my blocks` help keep code organised in the **Nature rover project** (https://projects.raspberrypi.org/en/projects/nature-rover).

`My blocks` also help by making it so you don't have to write the same code over and over again. You can use `my blocks` for the **Rocks** to position them in the spaceship's port.

Create a new `block` and call it `go to position`. The starting position can be anywhere on the screen.

The **Rocks** sprite needs to keep finding a random position until it is touching the **port** `and` not touching the edge of the **port**. This is similar to the code you used on the **crosshair**, but this time you will use an `and` block.

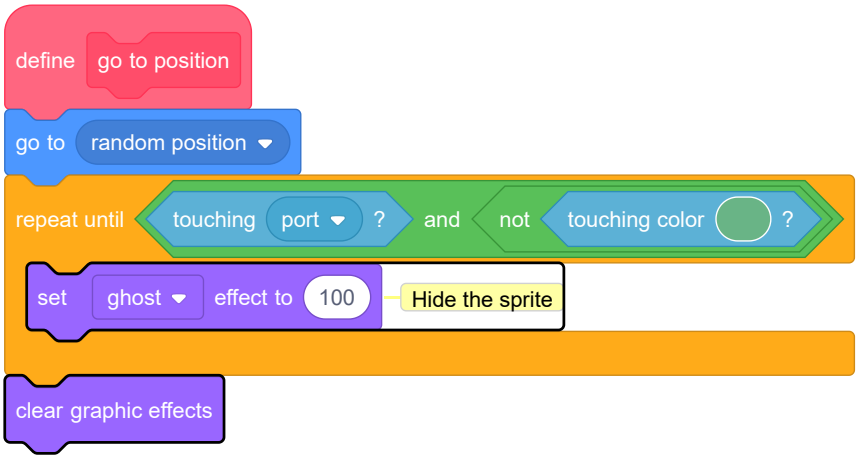
Add a `repeat until` and an `and` block to make sure that the **Rocks** keep moving until they are in the correct position.



Test: Click on your `my blocks` definition and you should see the rock move randomly around the screen, until it stops in the port.

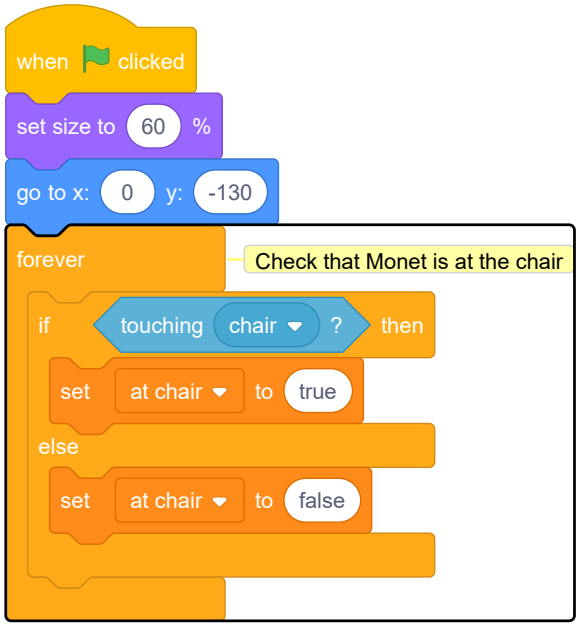
The **Rocks** need to be hidden from view as they move, but if they are hidden, they won't be touching the **port**, so a `ghost` effect can be used to make them invisible.

Set the `ghost` effect on the **Rocks** to `100` while the sprite is moving, and then clear the graphical effect.

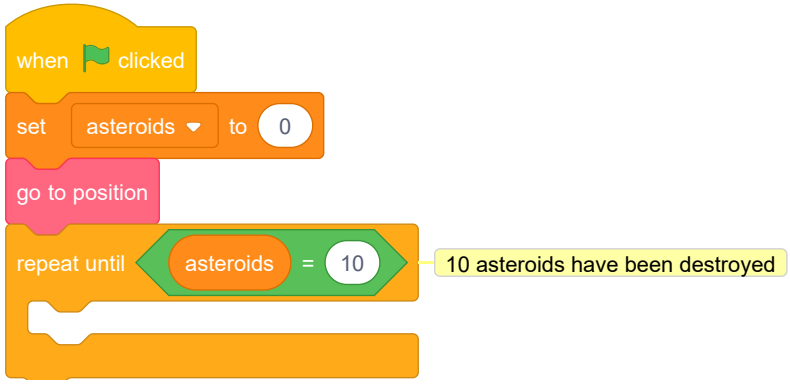


The last part is similar to the other puzzles. Using a variable called `asteroids`, count how often the **crosshair** touches the **Rocks**. Each time it does, the asteroid variable increases and the **Rocks** move to a new position. This should only work if your character is at the chair though.

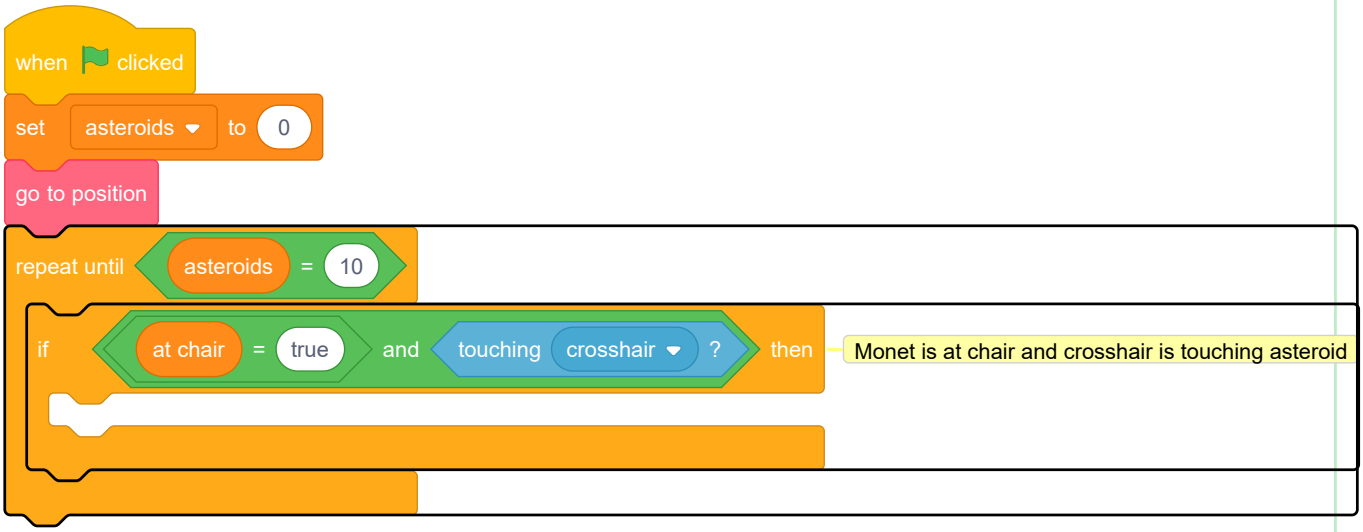
On your **character** sprite, create a new variable called `at chair`, and set it to `true` when the character is touching the chair and `false` when they are not.



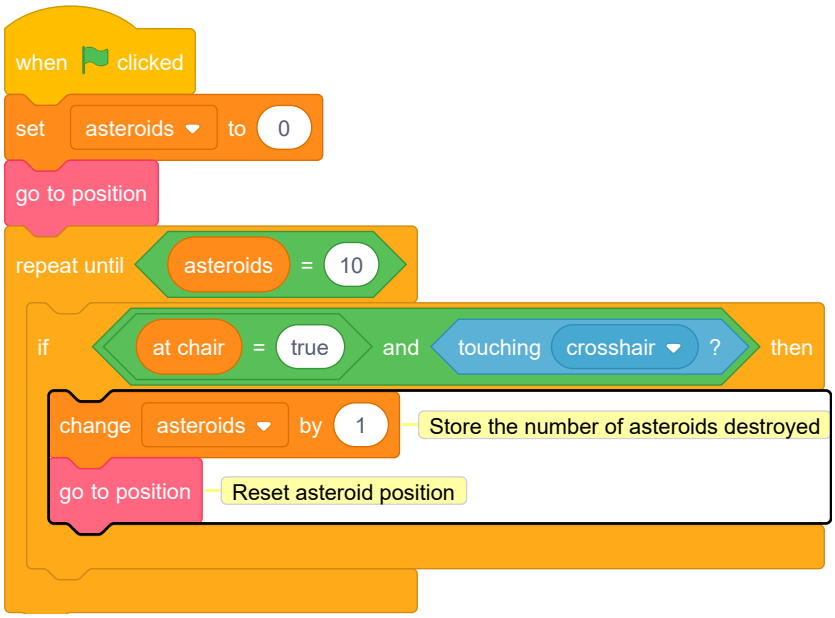
Add the following blocks to the **Rocks** sprite to set when the task is complete.



Use another `and` block in an `if` block to check that the **crosshair** is touching the rock and that the `at chair` variable is `true`.

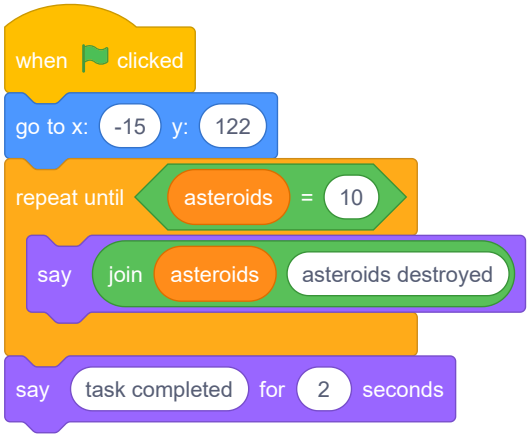


If the condition has been met, then the `asteroids` variable can be increased by `1` and the `my block` can be called again so the **Rocks** move to a new position.



The last thing to do is tell the player about the task. This can be done on the **port** sprite.

Add blocks to tell the player how many asteroids have been destroyed.



Test: Move your character close to the chair, then move the crosshair around and try to destroy some asteroids. You can then adjust any of the values in your code that you need to in order to make it work well for your sprite sizes.

Save your project

Upgrade your project

If you have time, you can upgrade your project.

Here are some ideas you could try:

- Add in extra sprites to provide more puzzles
- Edit the backdrop and turn areas into new sprites
- Add an extra challenge, by making the player complete all the challenges in a set time

You could use this **lever image** (https://projects-static.raspberrypi.org/projects/puzzle-room/f10166df7a3f68d591ceb1cffea59b4f6f35162e/en/images/lever.png) as a new sprite, and this **new backdrop image** (https://projects-static.raspberrypi.org/projects/puzzle-room/f10166df7a3f68d591ceb1cffea59b4f6f35162e/en/images/upgrade-backdrop.png) with the bottom right levers removed to create a new task. Both are used in the project shown below.

Puzzle room upgrade: **See inside** (https://scratch.mit.edu/projects/540387423/editor)

Save your project

What next?

If you are following the **Further Scratch** (https://projects.raspberrypi.org/en/pathways/further-scratch) pathway, you can move on to the **Mandala** (https://projects.raspberrypi.org/en/projects/mandala) project. In this project, you will make computer generated mandalas.

If you want to have more fun exploring Scratch, then you could try out any of **these projects** (https://projects.raspberrypi.org/en/projects?software%5B%5D=scratch&curriculum%5B%5D=%201).
