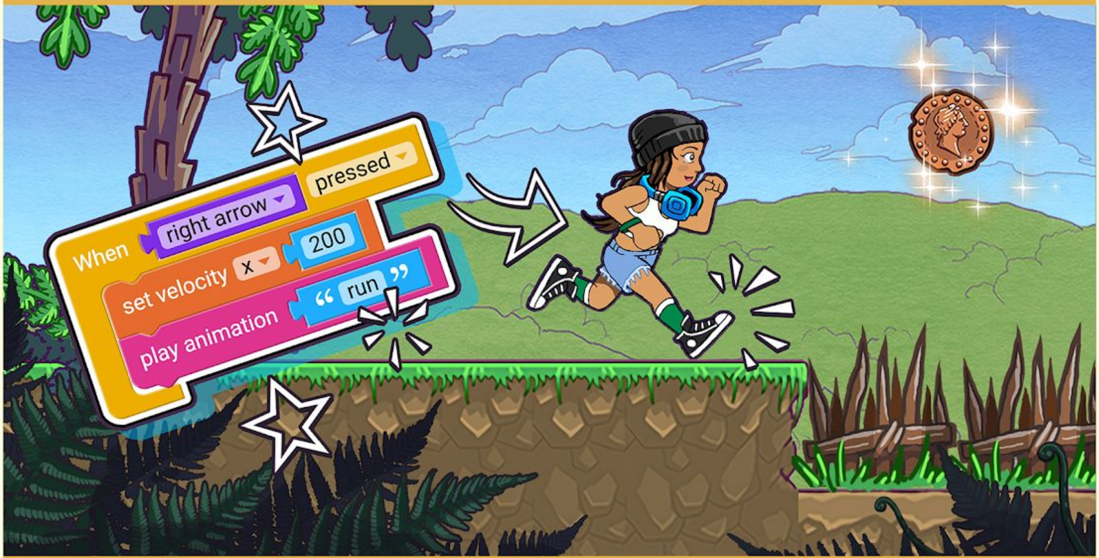
TE HIKO TĀKARO

How to code a 2D platformer game



Book 1 of 2

Develop a video game and learn to code

Year 7-10

Dan Milward and Gerard MacManus

In Gamefroot you can create games, interactive stories and animations and share them with others around the world.

Gamefroot is an online game creation platform that runs on laptop and tablet browsers.

This booklet and the Te Hiko Tākaro programme were developed by Gamefroot, in association with Te Puni Kōkiri: the ministry of Māori Development, Pātaka Art + Museum, and the Ministry of Education.

The learning outcomes in this booklet align with the New Zealand Curriculum (Digital Technologies learning areas). Completing this booklet is worth over 6 NCEA credits!

GETTING STARTED IN GAMEFROOT

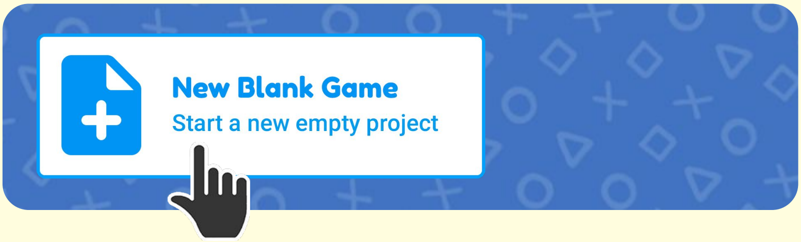
With the help of this booklet, we are going to make a platformer game like Super Mario Bros, and publish it online!

To get started head to the Gamefroot website by using the link below, open up the Editor, and get everything set up.

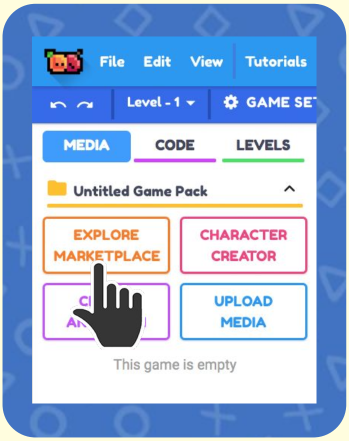
https://make.gamefroot.com/

Step 1: Gamefroot

When you sign in with your account, Gamefroot will take you to the Templates page. From there, select New Blank Game. This opens up the Game Editor.



In Gamefroot you can make just about any 2D game you can think of. Any time you want to create a new game, just go to the Gamefroot website and select Blank Game.

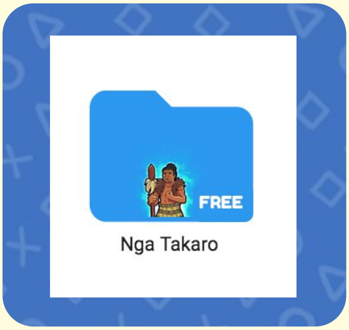


Step 2: Game assets

Now that you are in the Level Editor, let's grab an Asset Pack from the Marketplace.

Packs contain collections of characters, objects, artwork and scripts that you can use to build games.

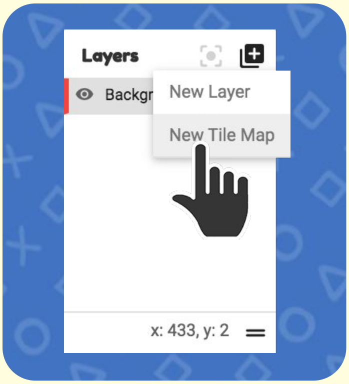
Click the **Marketplace icon** on the very left of your screen, as seen on the diagram to the left.



Step 3: 'Nga Tākaro' Pack

Click to open the Marketplace, and find the **Nga Tākaro** pack.

Click on the pack. On the next screen, click **Get it for Free** in the bottom right. This loads all of the pack's assets into your game.



Step 4: Using Layers

Find the Layers panel. Click on the **New Layer** icon and select **New Tile Map.**

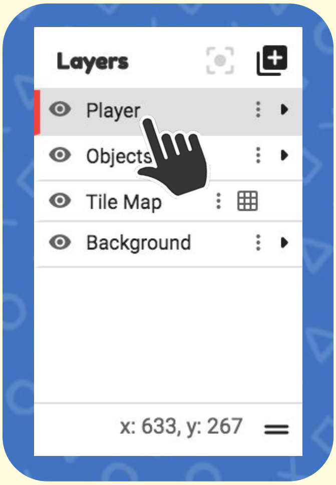
Tile assets such as platforms and water tiles can only be drawn on tilemap layers.

Step 5: More Layers!

After creating the Tilemap layer, create **two** new layers.

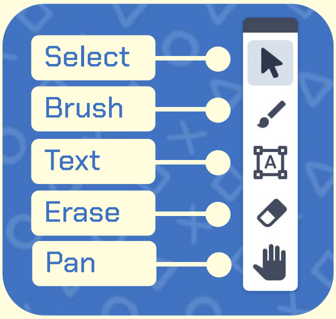
The top layer will be for the player character. The bottom layer will be for all the objects in the level such as trees, obstacles, and items. The Background layer, which already exists, is for background images and scenery.

Double-click on a layer to rename it.



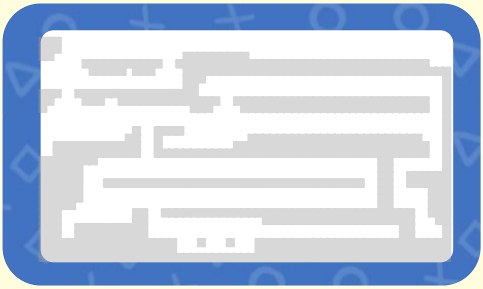
Step 6: The Toolbar

This is the Toolbar, consisting of Select, Brush, Erase, and Pan. Use Select to move, rotate, and scale assets, and use Pan to move your view around the level. Next, you will set up a Tracer Map and use it as a guide to help design your first level.



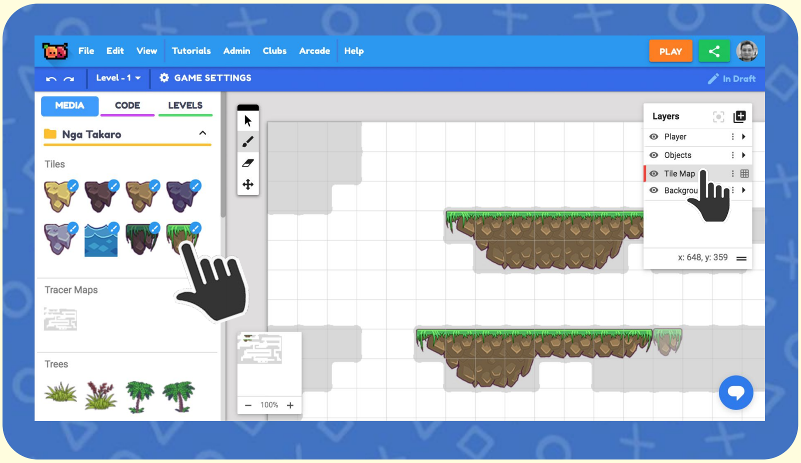
Step 7: Tracer Maps

Make sure that the Background layer is selected. Find the Tracer Map in the Game Objects panel. It will look like the grey block image above. Click on it and place it within your level.



Step 8: Using Layers

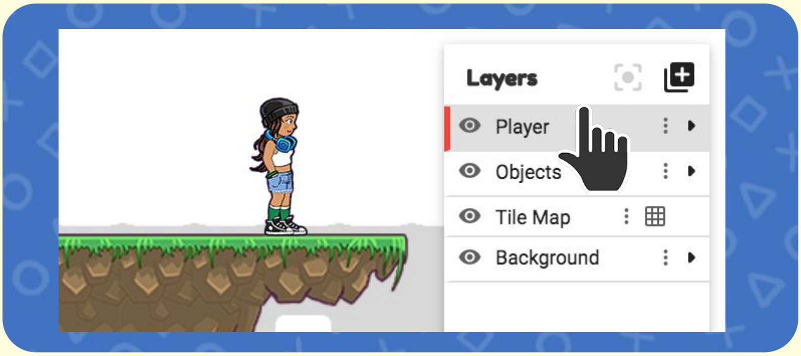
Switch to the **Tile Map layer** and select the grassy tile brush from the Game Objects panel. Use this brush to create a series of platforms by clicking and dragging across the tilemap.



Step 9: Playable Characters

Now, select a playable character for the player to control. Find a playable character in the **Game Objects panel.** Choose either the girl or the boy character.

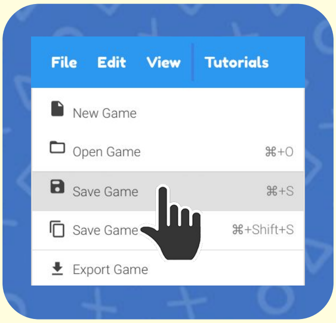
Select the **Player layer** then place your chosen character into the level.



Step 10: Save your progress

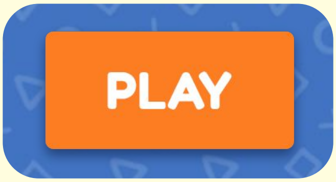
Now, it is time to save your game and test it out.

To save, click **File** on the Menu Bar and select **Save Game.**



Step 11: Playtesting

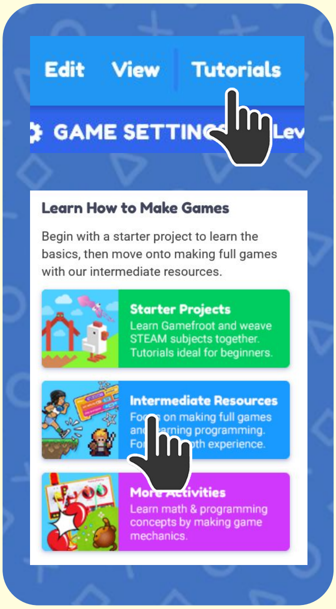
To test or play your game, just hit the big orange **PLAY** button in the top right!



Step 12: Tutorial sidebar

This tutorial, along with many others, are available in the **Tutorials** sidebar.

Click on the **Tutorials** button in the top menu bar, select the **Intermediate Resources**, and then open the **Platformer 101** tutorial.



Learn How to Make Games

Begin with a starter project to learn the basics, then move onto making full games with our intermediate resources.

Starter Projects
Learn Gamefroot and weave STEAM subjects together. Tutorials ideal for beginners.

Intermediate Resources
Focus on making full games and learning programming. For those with experience.

More Activities
Learn math & programming concepts by making game mechanics.

CODING CONVENTIONS

Coding conventions are style guidelines for programming. They typically cover:

- Naming and declaration rules for variables and functions
- Rules for the use of spacing and comments
- Programming practices and principles

Coding conventions

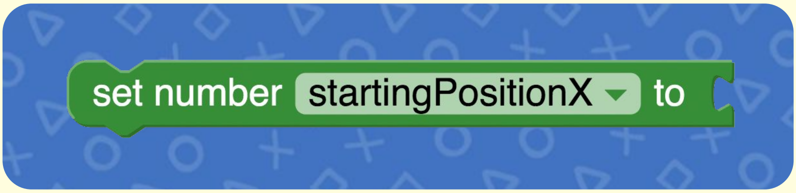
- Secure quality
- Improve code readability
- Makes code maintenance easier

Naming conventions

When making games in Gamefroot, we should follow widespread JavaScript naming conventions. These allow us to distinguish variables, classes, and constants, so we can more easily see what code is supposed to do.

Variables

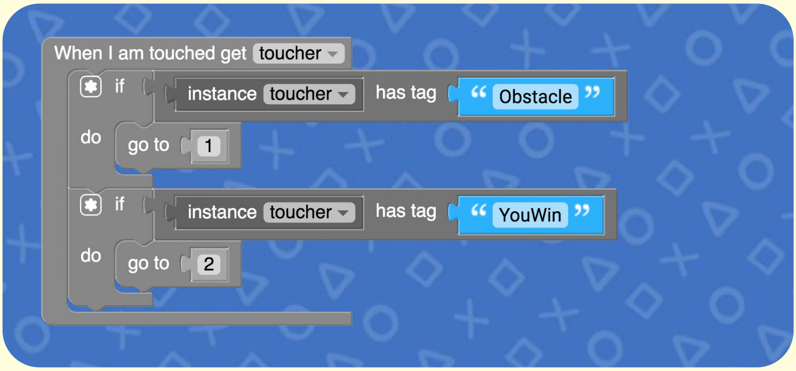
Variables are places to store changeable data. Name them in Camel Case. Start by making all capital letters lowercase. Then capitalise the first letter after each space. Finally, take out all the spaces. For example **starting position x** becomes **startingPositionX**.



```
set number startingPositionX ▾ to
```

CLASSES and UNIQUE IDENTIFIERS

A class is a specific kind of object that can be distinguished from other objects, like a name or type. For example, two obstacles have the same class. Name them in Pascal Case. This is just like Camel Case, but the first letter is capitalised too. For example, **you win** becomes **YouWin**.



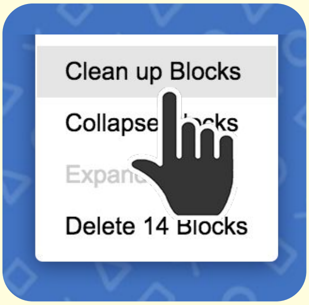
```
When I am touched get toucher
    if  instance toucher has tag "Obstacle"
    do  go to 1
    if  instance toucher has tag "YouWin"
    do  go to 2
```

CONSTANTS

Constants are places to store data that never changes. Name them in Screaming Snake Case. Start by making all letters capitals. Then replace all spaces with underscores. For example, **starting position x** becomes **STARTING_POSITION_X**. Constants look distinctive, so you can quickly tell if an item is supposed to change.

CODE FORMATTING

To tidy code in Gamefroot, right-click on the Script Stage and click **Clean up Blocks**. This makes it much easier to find code in a large script.

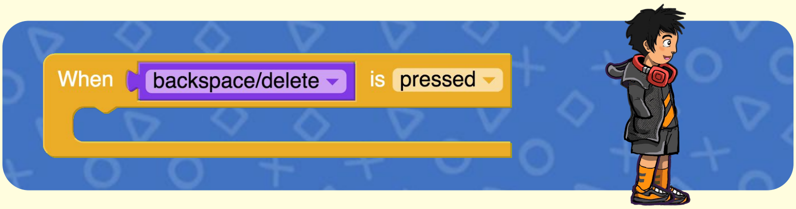


CODING PLAYER CONTROLS WITH INPUTS AND OUTPUTS

In this chapter, you will add a script to the player and program it to move around the level. To achieve this, you will use inputs and outputs, event-driven programming, sequencing, and iteration, also known as loops.

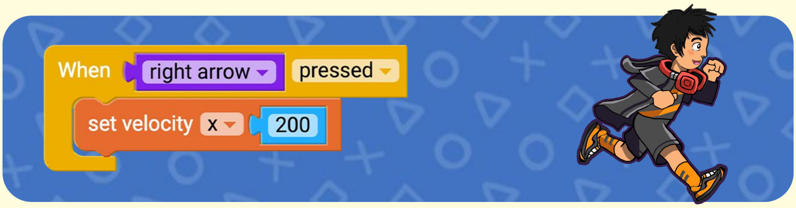
Adding a player script

Once you have set up your game, right-click on the player character and click **Add Script**. Open the Events palette on the left. Find the **When backspace/delete pressed** block and drag it out into the workspace. Change **backspace/delete to right arrow.**



```
When [backspace/delete ▾] is [pressed ▾]
```

Find **set velocity x 0** in Physics and drag it into the **When backspace/delete pressed** block. Change **0** to **200.**



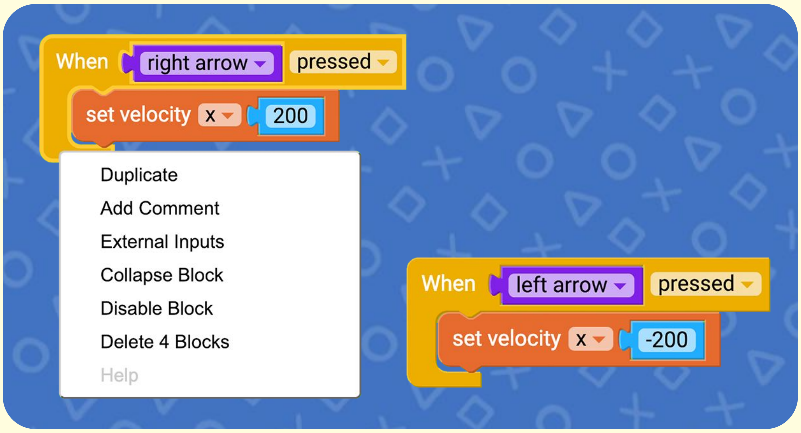
```
When [right arrow ▾] [pressed ▾]
    set velocity [x ▾] (200)
```

Now, hit the **PLAY** button and test your game. The character will move to the right when you press the **right arrow key.**

After you are done testing, close the game and return to the Script Editor.

So far, your character can only move to the right. Let's add the ability to move in other directions.

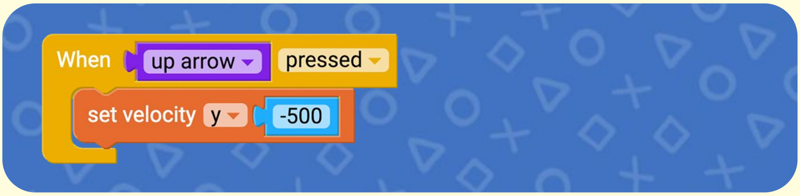
Right-click and duplicate the **When right arrow pressed** block. Change **right arrow** to **left arrow** and velocity from **200** to **-200** on the new block.



```
When [right arrow] [pressed]
    set velocity [x] [200]
```

```
When [left arrow] [pressed]
    set velocity [x] [-200]
```

Duplicate the block again and this time, change the new block to **up arrow**. Next, change velocity from **x** to **y** and the number to **-500.**



```
When [up arrow] [pressed]
    set velocity [y] [-500]
```

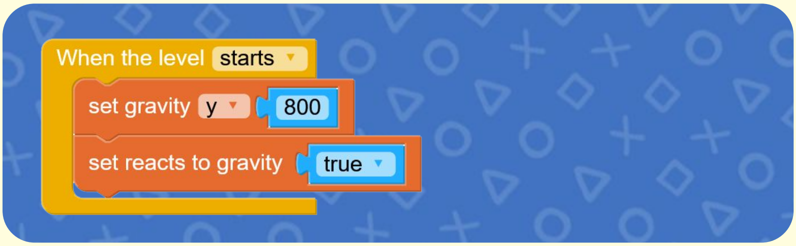
Hit **PLAY** and test moving left and up. See what happens? The direction keys work but the player doesn't stop moving.

To fix this, you need to add gravity to the game, and program the player character to stop moving when a direction key is released.

Adding Gravity

Let's code Gravity to start working as soon as the level starts. Grab a **When the level starts** block from the **Events** palette. Drag **Set gravity x 0** into it from **Physics.** Change **x** to **y** and **0** to **800.**

By default, objects won't react to gravity. Grab **Set reacts to gravity true** block from **Physics.** Place it under **Set gravity y to 800.** This ensures that the player character will fall back to the ground after jumping up.

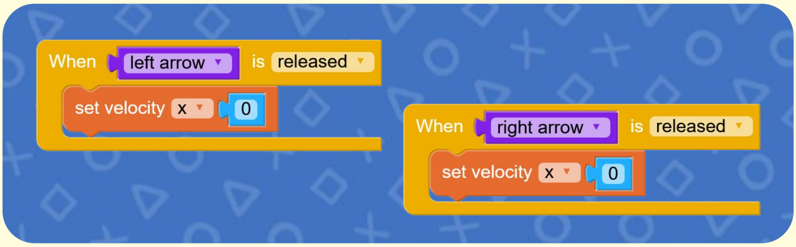


```
When the level starts
    set gravity y 800
    set reacts to gravity true
```

The next step is to detect when a key is released. Let's program ours so that the character will stop moving when the direction key is released.

Duplicate the **When left arrow pressed** block, change **pressed** to **released**, and set the velocity to **0.**

Follow the same process to make the player stop moving when the right key is released.



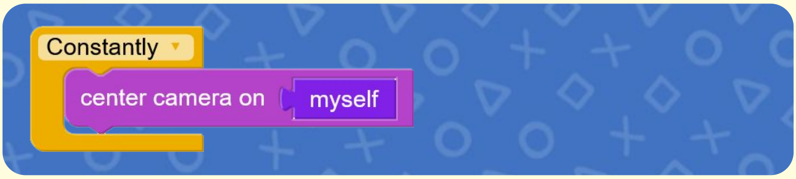
```
When left arrow is released
    set velocity x 0

When right arrow is released
    set velocity x 0
```

Game Cameras

Next let's code a camera to follow the player as you move around the game. This means that as you move around the level, you won't ever disappear off the side of the screen.

To do this, grab the **Constantly** block from Events. Then connect the **center camera on myself** block from Looks.



```
Constantly
    center camera on myself
```

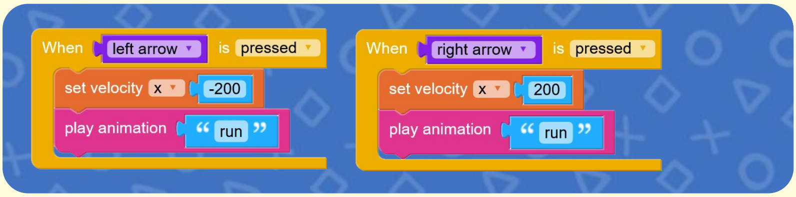
Test your game by hitting **PLAY.** Use the arrow keys to check everything we've built is working

Animating the player

The final step of Chapter 1 is to make the player character animate when they move around.

Each character in the Nga Tākaro pack has animation states for running, jumping, climbing and standing still (idle). Let's make the player character run when the left and right arrow keys are pressed.

Grab the **play animation name** block from the Animation palette, and connect it into your **When right arrow pressed** block. Change the name to **run**. This will trigger the running animation to play. Grab the same block and put it in **When left arrow pressed** as well.



```
When left arrow is pressed
    set velocity x -200
    play animation "run"
```

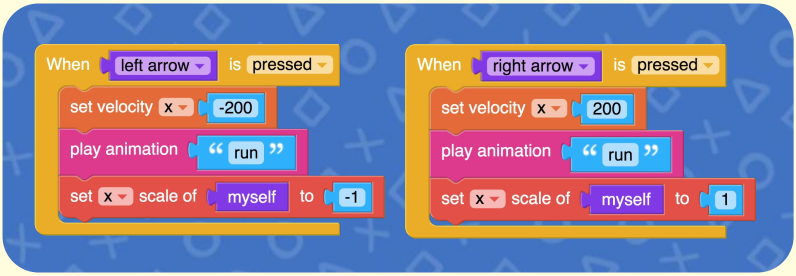
```
When right arrow is pressed
    set velocity x 200
    play animation "run"
```

To make the player go back to standing still, you need to trigger the players **idle** animation. To do that, put **play animation name** blocks into **When left arrow released** and **When right arrow released**. Change the animation names to **idle**.

Flipping the player

Finally, let's flip the player to face left and right depending on the key you press. Open the **Transform** palette and drag a **set scale x of myself to 1 block inside When left arrow pressed.**

Open the **Transform** palette again and drag a **set scale x of myself to 1 block** inside **When right arrow pressed** script. Change **1** to **-1**. This will scale (resize) the player on the x axis (horizontally) to make them appear flipped.



```
When left arrow is pressed
    set velocity x -200
    play animation "run"
    set x scale of myself to -1
```

```
When right arrow is pressed
    set velocity x 200
    play animation "run"
    set x scale of myself to 1
```

Test your game by hitting **PLAY.** Run around and make sure the player animates nicely.

Save your game instructions

Great work! You've learnt how to use:

☐ Inputs & outputs ☐ Sequencing

☐ Event-driven programming ☐ Iteration (loops)

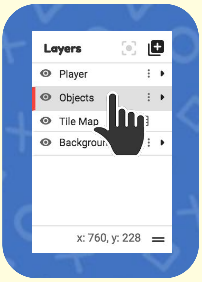
Confirmed by ______________________

CODING COLLISION DETECTION

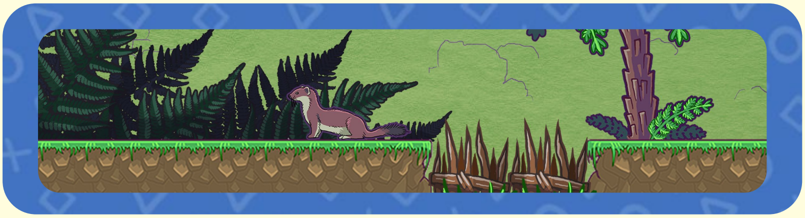
In video games, a collision happens when two objects intersect or when the distance between those objects falls below a certain tolerance. In this chapter, you are going to edit the player script so that it will detect when it collides with other objects.

Collision Detection

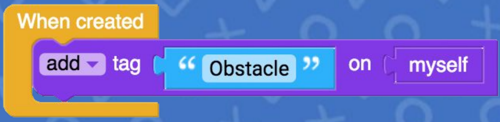
Objects will be converted to tiles if placed on a tilemap layer. To avoid this, make sure that the Objects layer is selected in the Layers panel.



Place some objects onto the level for the player to collide with. In our example we are using the wooden spikes.



Right-click on one of the obstacles and click **Add Script.** Open Events and drag out a **When created** block. Then open Sensing and drag a **Add tag tag name on myself** block inside it. Change tag name to Obstacle and close the Script Editor. Save the script as Obstacle.



When created
add tag "Obstacle" on myself

Now, let's add this script to all of your obstacles. You can reuse this script on any object that you want to be collidable.

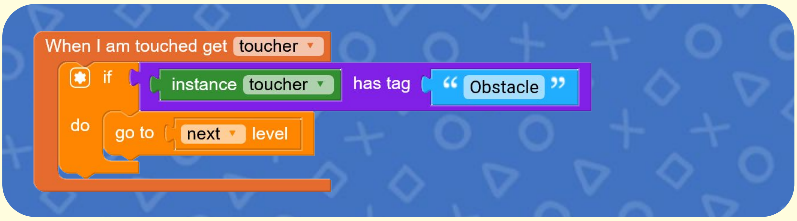
To do this, open the Code sidebar, and select your **Obstacle** script and click on any object in the level. This will attach the new script and turn the object into an obstacle.

Next, let's code the player to detect when it collides with another object and restart the level.

Right-click on the player character and click Edit Script. Open the Physics palette. Drag a **When I am touched get toucher** block into the workspace. Click on Control Flow. Drag an **if do** block inside the **When I am touched get toucher** block. This block will allow us to check if a condition is true. . .

Now we want to check if the object the player touched was tagged as an obstacle. Click on the Sensing palette and drag a **myself has tag name** block into the empty **if** slot. Open Variables, drag a instance toucher block onto the myself block to replace it, delete the myself block that pops out. Change the tag name to Obstacle.

If the instance toucher is an obstacle, we will make you lose by restarting the level. Finally, grab **go to next level** from Control Flow, and change **next** to **current.**



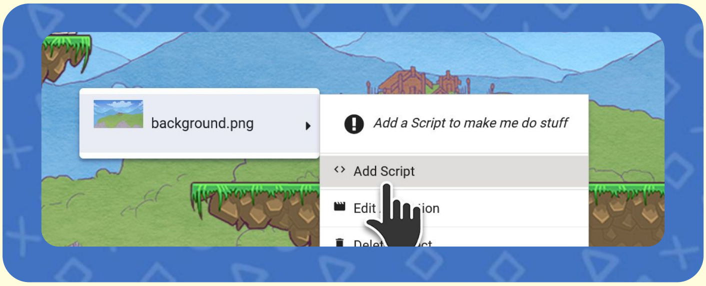
```
When I am touched get toucher
  if   instance toucher  has tag  "Obstacle"
  do   go to  next  level
```

Hit **PLAY** to test what we have built so far. The game should restart every time the player collides with an obstacle.

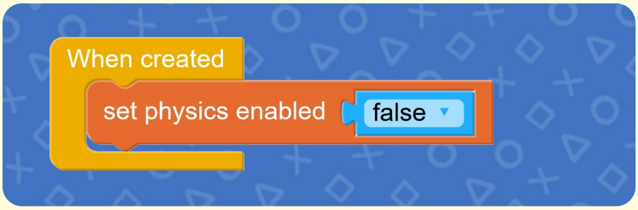
Non collidable objects

Sometimes we do not want the player to collide with an object, such as background images. In this step we are going to code our game's background to be non-collidable.

Start by selecting the **Background layer.** Find a background image from Media sidebar and place it on this layer. Right-click on your image and click Add Script.



Drag a **When created** block from the Events palette into your workspace, then drag **set physics enabled true** block from Physics into it. Change **true** to **false.**

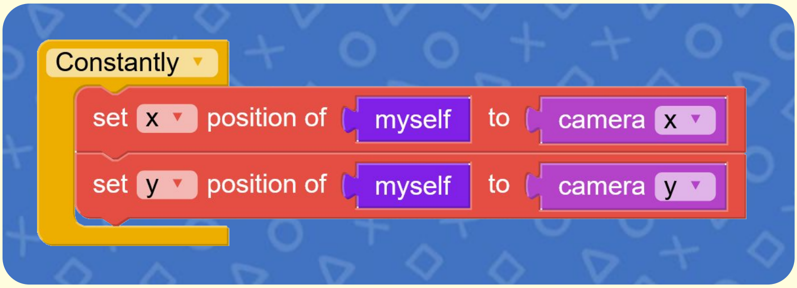


The second half of our script will lock the background image in place so that it looks the same no matter where the player goes.

Click on the Events palette and drag a **Constantly** block into your workspace. Open the Transform palette and find a **set x position of myself to 0** block and place it inside **Constantly.**

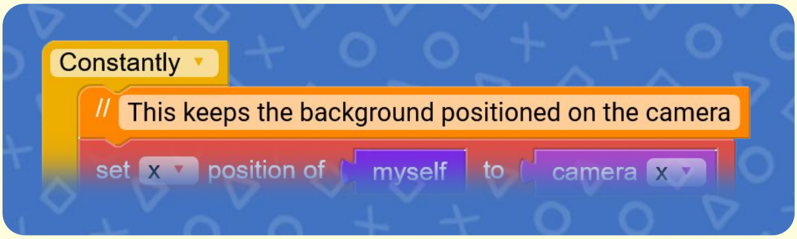
Open Looks and drag a **camera x** block onto the **0** block. This will pop out the **0** block which you can delete.

Right click and Duplicate the **set x position of myself to camera x** blocks, connect the duplicated blocks under the **set x position of myself to camera x** and change both **x** values to **y.**



```
Constantly
    set x position of myself to camera x
    set y position of myself to camera y
```

Lastly, you should add a **code comment** to describe what your code is supposed to do. Open Control Flow, drag an **// Add Comment** block into the Constantly block, and then type out a description of what your code is supposed to do.



```
Constantly
    // This keeps the background positioned on the camera
    set x position of myself to camera x
```

Code comments are useful for when you come back to your code later, or when someone else is reading over your code so that they can easily understand it.

Great work! You've learnt how to use:

☐ Selections with conditional logic ☐ If statements

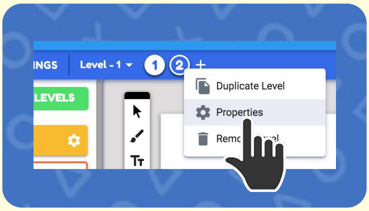
Confirmed by ______________________

LEVEL NAVIGATION

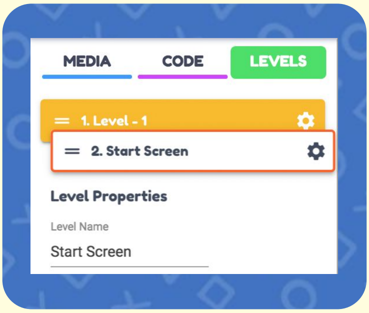
In Gamefroot, game levels are like chapters in a book. Players work their way through one, and when finished, move on to the next. In this chapter, you will add levels to your game and learn how to navigate between them.

Levels

Start by adding a new level to your game. To do this, click the **+** button, which will add a second level. Then right-click on level **2** and click the **Properties** button. This will open the levels sidebar.



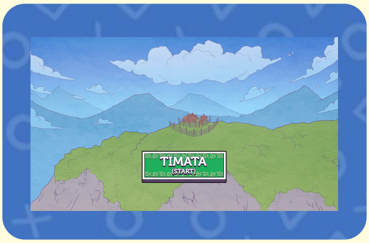
In the Levels sidebar, rename Level 2 to **Start Screen.**



Level Name: Start Screen

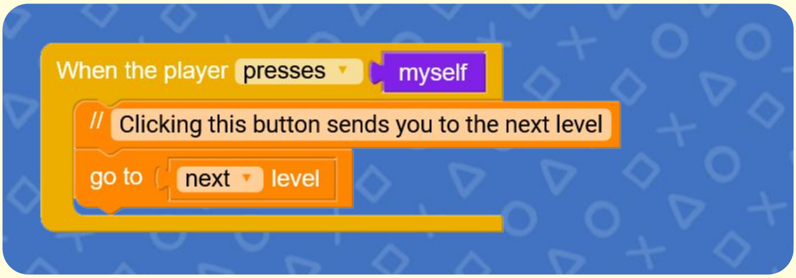
Drag this level up above Level 1 so that Start Screen becomes the first level.

Open the Start Screen level by clicking on it. In the sidebar, click **Media** and find objects to fill up your empty Start Screen. Include a **Timata (Start) button.**



The Timata button will be what you press to start the game.

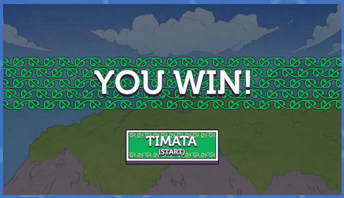
Grab a **When the player presses myself** block from the Events palette and put it in your workspace. Open Control Flow and drag a **go to next level** block inside it. Then save and close this script.

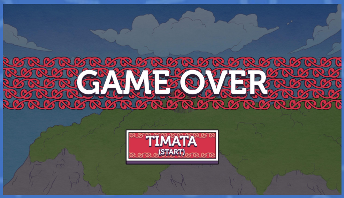


```
When the player [presses ▾] [myself]
    // Clicking this button sends you to the next level
    go to [next ▾] level
```

Now add two more levels. Open the Levels sidebar and rename your third level to **You win**, and rename your fourth level to **You lose.**

Add objects and decorations to level 3 and level 4 so that each level looks like a proper **You win** and **You lose** screen.





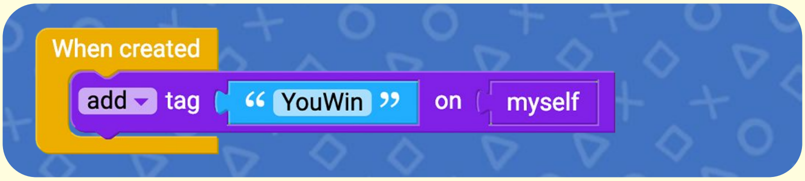
Your levels should now be arranged like this:

- Start screen
- Level - 1 (your playable level)
- You Win screen
- You Lose screen

Next, let's add a trophy object to the game to take the player to the **You win** level.

Switch back to the playable level, select an object from the Media sidebar and put it in your level.

Add a new script to this object. Drag out a **When created** block from Events and put an **add tag tag name on myself** block inside it (from Sensing). Change **tag name** to **YouWin.**

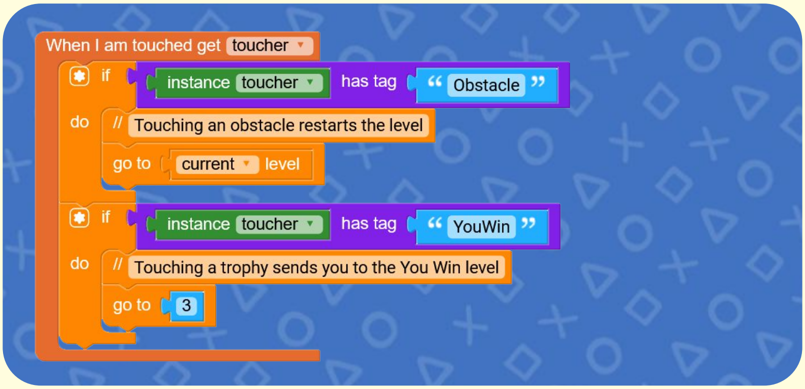


```
When created
  add ▾ tag "YouWin" on myself
```

Now, let's make the player detect when it collides with the trophy. Open the player script find your existing collision detection code.

Right-click on the **if do** block and click Duplicate. Connect the new blocks underneath the first **do** block.

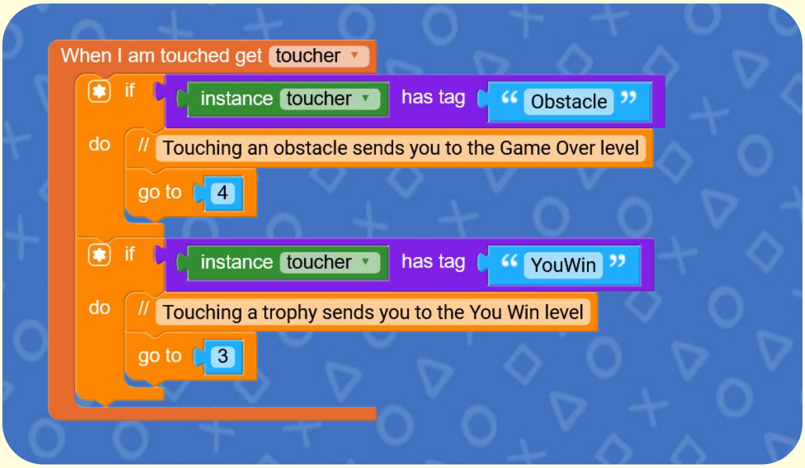
Click on Operators and drag a **0** block on to the **next level** block. This will pop the block out which you can delete. Change 0 to 3 (the **You win** level). Your code should look like this:



```
When I am touched get toucher ▾
  if instance toucher ▾ has tag "Obstacle"
  do // Touching an obstacle restarts the level
     go to current ▾ level
  if instance toucher ▾ has tag "YouWin"
  do // Touching a trophy sends you to the You Win level
     go to 3
```

Now let's tell the collision detection code to send the player to the **You lose** screen instead of resetting the level when they collide with an Obstacle.

Drag out the **current level** block from **go to current level** block and delete it. Open Operators and drag a **0** block into the empty **go to** block and change the number to reflect what your game's **You lose** level number is. Your code should look like this.



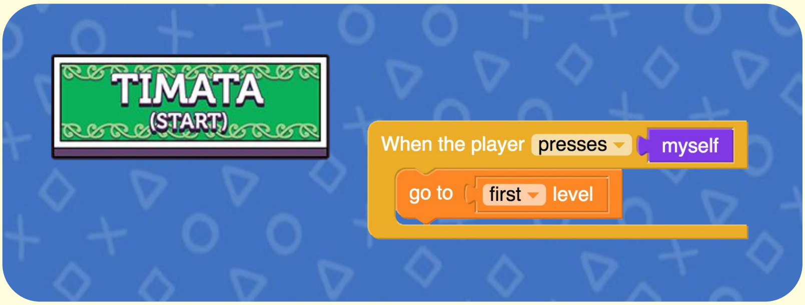
```
When I am touched get toucher
    if  instance toucher  has tag "Obstacle"
    do  // Touching an obstacle sends you to the Game Over level
        go to 4
    if  instance toucher  has tag "YouWin"
    do  // Touching a trophy sends you to the You Win level
        go to 3
```

In the next step, we will create a Replay button so that if the player loses they can play your game again.

Open the Media Sidebar and find the Timata button. Add the button to both the You win and the You lose levels.

Add a new script to one of these buttons. Open Events drag a **When the player presses myself** block into your workspace. Open Control Flow, drag a **Go to next level** inside the **When the player presses myself** block. Change **next** to **first** and save the script.

Open the Code sidebar and add this new script to the other Timata button.



```
When the player presses myself
  go to first level
```

Great work on Chapter 3! Make sure to save your work.
If everything is done right, the player should now see a **Start screen** when they load the game, a **You lose** screen when they die, and a **You win** screen when they collect the trophy.

Great work! You've learnt how to use:

- ☐ Conditional logic
- ☐ If statements
- ☐ Mouse input events

Confirmed by ____________________

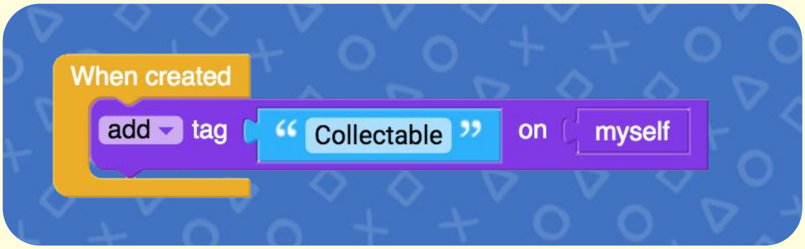
KEEPING SCORE WITH VARIABLES

In this chapter, you will create collectable objects that increase the player's score on collision. You will also edit the player script and implement number variables to create a basic scoring system.

Variables

Let's start by adding some more objects to your level. This time choose objects for the player to collect. We use a coin in our example.

Add a new script to one of them. Open Events and drag a **When created** block in your workspace. Open Sensing. Drag an **add tag tag name on myself** block in to the **When created** block and change **tag name** to **Collectable.**

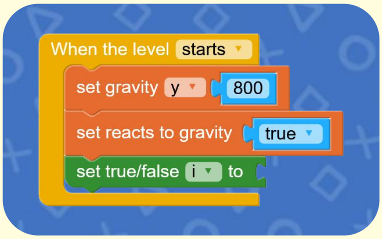


```
When created
  add ▾ tag " Collectable " on myself
```

Close and save the script. Open the Scripts panel and add your new script to all the collectable objects.

Now let's create a *number* variable to store the players score. You're going to code the score to increase every time the player touches one of the collectable objects.

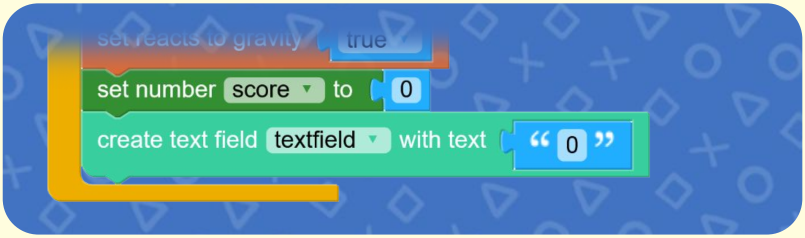
Now open the player script. Open Variables, and drag **set true/false i** to the bottom of the **When the level starts** block.



```
When the level starts
    set gravity y to 800
    set reacts to gravity true
    set true/false i to
```

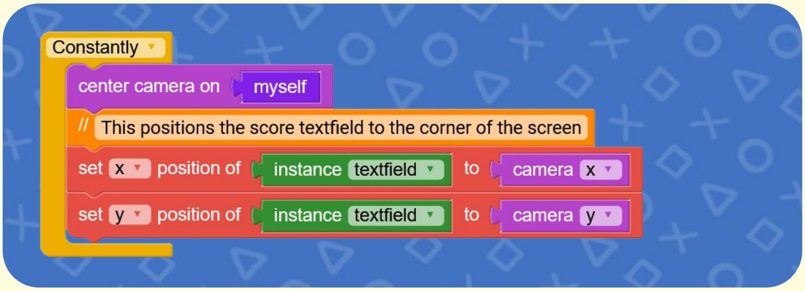
Click on **i,** select **new variable** from the list, name the variable **Score** and change **true/false** to **number.** Finally, grab a **0** block from Operators and put it in the blank space next to **score.**

Now grab a **create new textfield** from the Draw palette and connect it to the bottom of your script. Type **0** into the textfield.



```
set reacts to gravity true
set number score to 0
create text field textfield with text "0"
```

Finally, find a **set x position of myself to 0** block from Transform and place it inside your **Constantly** block. Replace the **myself** block with **instance textfield** from Variables and replace **0** with a **camera x** block from the Looks palette. Duplicate these blocks and change the two **x** values to **y.**

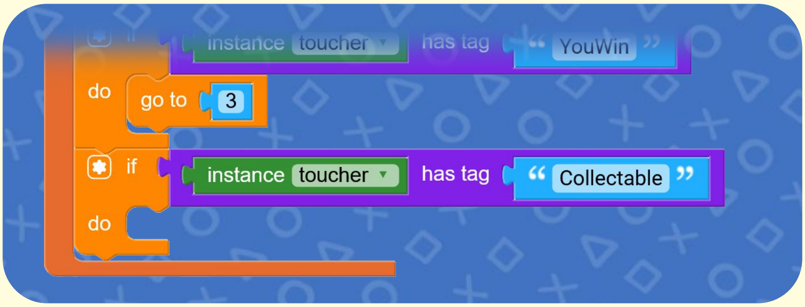


```
Constantly
    center camera on myself
    // This positions the score textfield to the corner of the screen
    set x position of instance textfield to camera x
    set y position of instance textfield to camera y
```

Close and save the script. Hit the PLAY button to see your score on-screen. It won't increase yet.

Now we'll make the score number increase when you pick up one of the collectables.

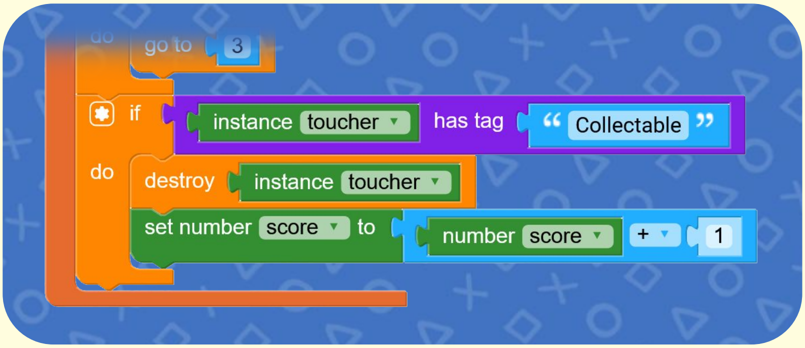
Inside **When I am touched, get toucher**, duplicate **if instance toucher has tag YouWin** (by right-clicking on it) and drag the duplicated block underneath the original block. Rename YouWin to **Collectable** and then delete **go to 3.**



```
if instance toucher has tag "YouWin"
do  go to 3
if instance toucher has tag "Collectable"
do
```

To pick up the collectable object, open Control Flow and drag **destroy myself** inside the if do block. Delete **myself** and replace it with **instance toucher** (from Variables).

To increase the score number, open Variables and drag **set number score to** underneath **destroy instance toucher.** Then open Operators and connect **1 + 1** into that block, and drag **number score** (from Variables) into the left side of the + block.



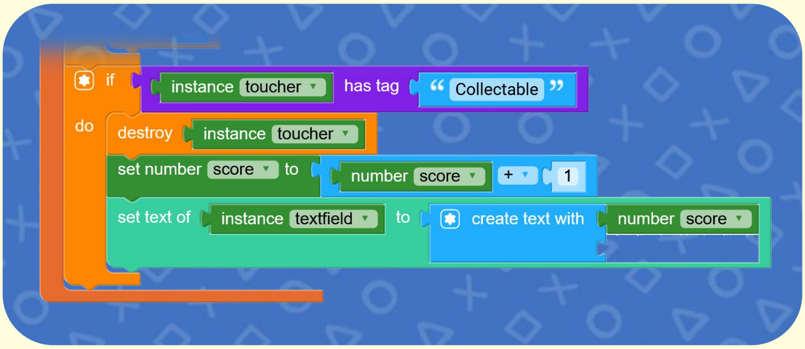
```
do  go to 3
if instance toucher has tag "Collectable"
do  destroy instance toucher
    set number score to number score + 1
```

Lastly, to update the textfield, open Draw and drag **Set text of instance textfield** underneath **set number score to score + 1.**

Now, grab the **number score** block and put it in the **set text of instance textfield to** block. But it won't connect! This is because it is a number value. The **set text of** block expects text, not a number. Click on number and change the value to text. Do this with the block **create text with** from Operators.

Connect the **create text with** block to the **set text of** block. Add the **number score** variable to one of the empty spaces.

Test out the game. If everything has been set up correctly, when the player collides with one of the collectable objects, the score number will increase and the collectable will disappear.



```
if  instance toucher  has tag  " Collectable "
do  destroy  instance toucher
    set number score to  number score + 1
    set text of  instance textfield  to  create text with  number score
```

Great work! You've learnt how to use:

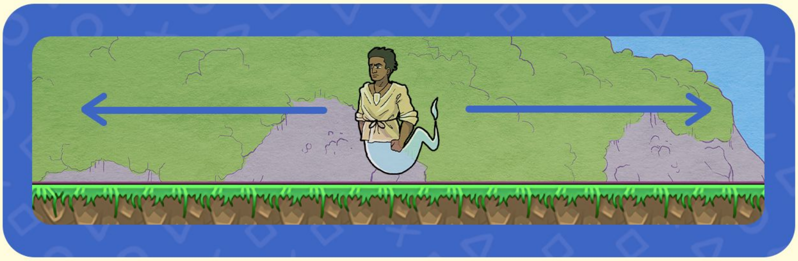
☐ Variables ☐ Data types

Confirmed by ____________________

CODING A MOVING OBSTACLE

In this chapter, you are going to code a Non-player character (NPC) to patrol back and forth from one place to another.

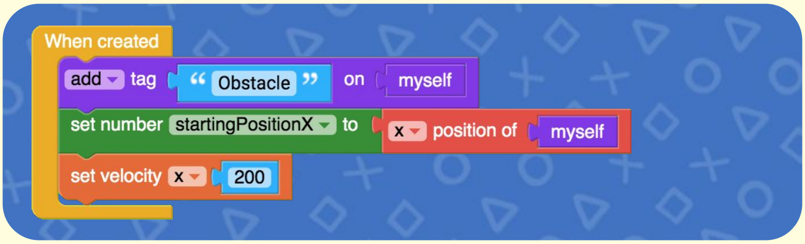
Place an object into your level that will become your patrolling character. Choose an area in your level with enough space on either side for the patroller to move around.



Add a new script to this object and drag out a **When created** block (from Events). Drag **Add tag** (from Sensing) into it and change **tag name** to **Obstacle.**

Add **Set true/false i** (from Variables) to this script block. Click on **i** and create a new variable with the name **startingPositionX.** Change **true/false** to **number.**

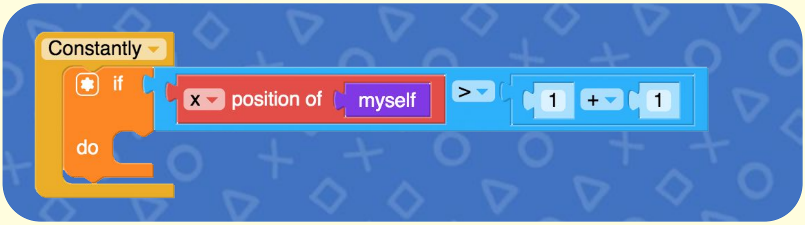
Add a **Set x position of myself** block (from Transform) into the **set number startingPositionX.** Add set **x velocity** (from Physics) below, and change **x** to **200**



```
When created
    add tag "Obstacle" on myself
    set number startingPositionX to x position of myself
    set velocity x 200
```

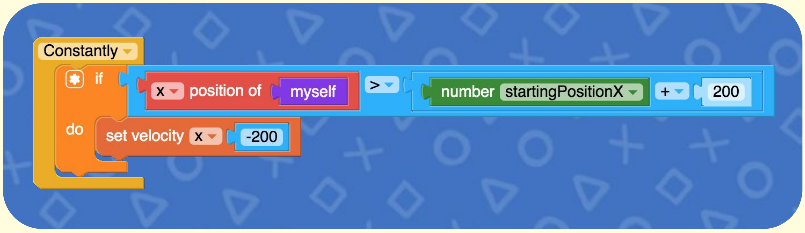
Next, we will stop the patrolling object wandering too far away from where it started. You will do this by comparing its current position to its original position, using a comparative operator..

Grab a **Constantly** block (from Events) and drag an **if do** block into it (from Control Flow). Add the **_=_** comparator to the **if** slot. This is in the Operators panel, in the Boolean section. Change it to **_>_** (greater than). Then add **x position of myself** (from Transform) into the empty space on the left, and **_+_** (from Operators) into the right of the **_>_**.


```
Constantly
  if  x position of myself  >  1 + 1
  do
```

Within this **_+_** block, add **number StartingPositionX** to the left, and **0** from Operators to the right. Change **0** to **200**, or however far you want the patroller to move.

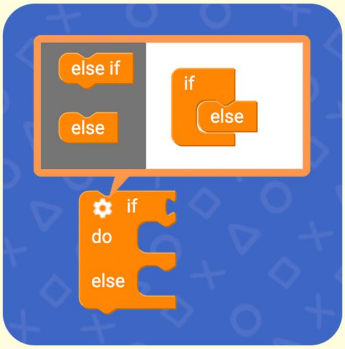
Underneath this, add Set **x velocity** to **0**, and change **0** to **-200.** Exit, save the script and hit **PLAY** to test things out.


```
Constantly
  if  x position of myself  >  number startingPositionX + 200
  do  set velocity x -200
```

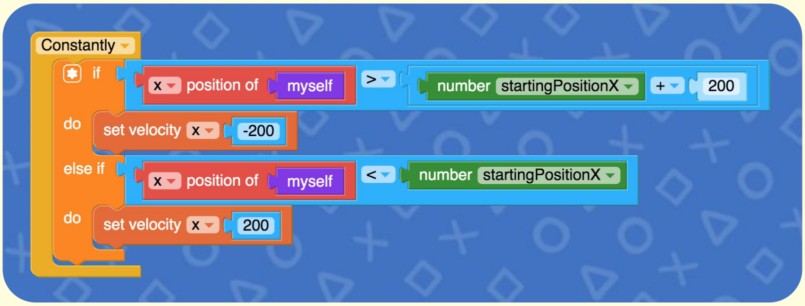
Next, we will make the patroller turn around when it has reached its maximum distance. This will make it move back and forth.

Start by modifying the **if** block we made. Change it to an **if else** if statement. This is used to program different actions depending upon whether the condition is true or false.

Click on the **if do** block's settings icon. On the pop-up, drag an **else if** into the **if** block. Click on the settings icon to close this pop-up.



Duplicate the **x position of myself > number startingPositionX + 200** blocks and connect them to the **else if** slot. Remove **startingPositionX + 200** from the duplicated blocks. Fill in the empty slot with the **startingPositionX** variable. Change **>** to **<** (less than).



```
Constantly
  if   x position of myself  >  number startingPositionX  +  200
  do   set velocity x  -200
  else if   x position of myself  <  number startingPositionX
  do   set velocity x  200
```

Duplicate set **velocity x -200**. Put it in the second do slot. Change -200 to **200.** Save the script and hit **PLAY** to test your game.

Great work! You've learnt how to use:

- ☐ Comparative operators
- ☐ Variables
- ☐ If-else selection control structures

Confirmed by ____________________

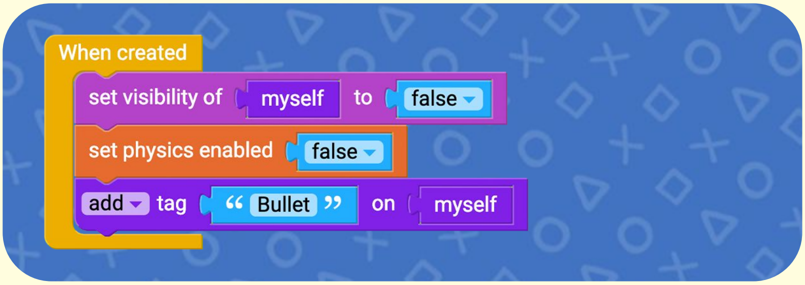
ADDING AND THROWING PROJECTILES

In this chapter, you are going to code an object for the player throw. You will use conditional logic to determine where to place the projectile, and to determine in which direction it needs to travel.

We will now expand our player character script to throw objects. The first thing we have to do is set up our projectile and get it ready to throw.

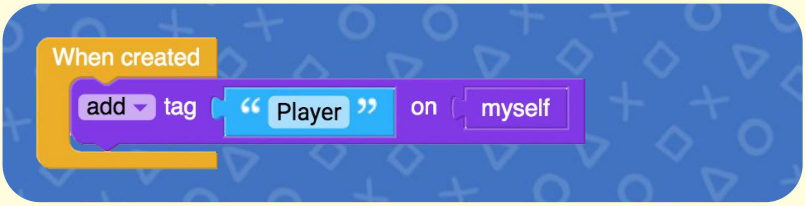
Place an object into your level that will become the throwable projectile. Right-click on this object to add a new script.

Drag in the **When created** block (from Events) and then drag in **set visibility true** (from Looks), **set physics enabled true** (from Physics), and **add tag tag name on myself** (from Sensing). Change both **true** values to **false**. Change **tag name** to **Bullet**. Then close and save the script.



```
When created
    set visibility of myself to false
    set physics enabled false
    add tag "Bullet" on myself
```

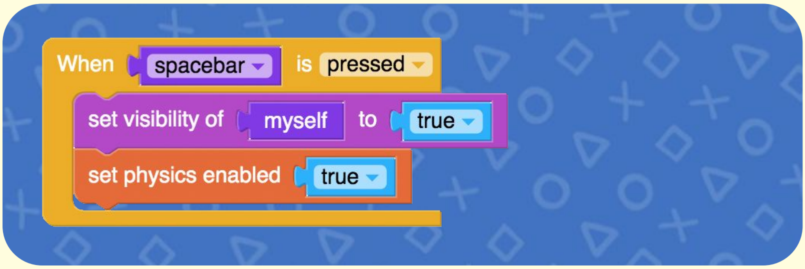
The projectile needs to be able to identify the player with a tag. Open up the player script. Drag out a **When created** block (from Events), and add **add tag tag name on myself** to it (from Sensing). Change **tag name** to **Player**.



```
When created
    add tag "Player" on myself
```

Next, we will make a keyboard trigger which makes the player character throw the projectile. We will use the Spacebar as our trigger. Open the projectile script again and add **When backspace/delete pressed** (from Events). Change **backspace/ delete** to **spacebar**.

Find and drag **set visibility true** (from Looks) and **set physics enabled true** (from Physics) into **when spacebar pressed**. Change both **false** values to **true.**

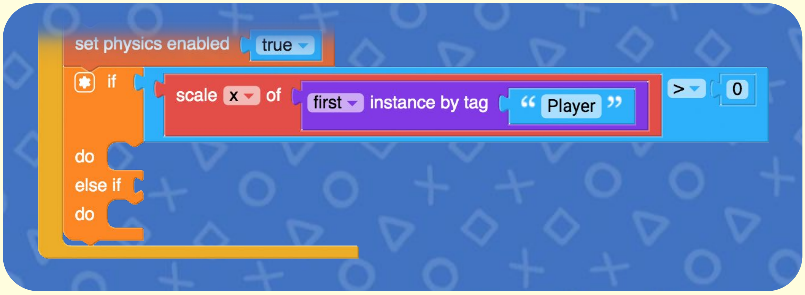


```
When spacebar is pressed
    set visibility of myself to true
    set physics enabled true
```

Now we'll make the projectile shoot left or right depending on the direction the player is facing. We'll use an if-else block to check the player's scale.

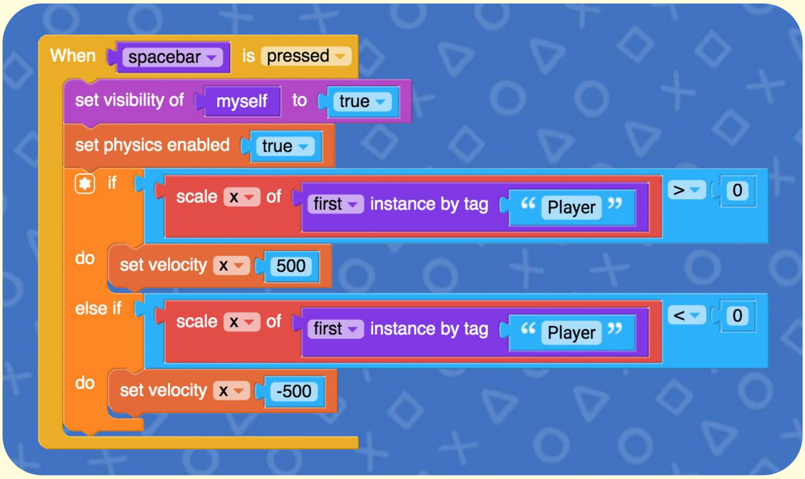
Grab an **if do** block (from Control Flow) and add it into **when spacebar pressed**. Click on its settings icon and change it to **an if else if** block.

Add a **_>_** (greater than) block to the first if slot. Place **scale x of myself** to the left and **0** to the right. Replace **myself** with **first instance by tag tag name**, and change **tag name** to **Player**.



```
set physics enabled [true]
if  scale [x] of [first] instance by tag "Player"  [>] 0
do
else if
do
```

Add a **set velocity x** block to the first **do**, and set it to **500**. Then duplicate both of these blocks and add them to the **else if** section. Change **>** of **_>_** to **<**, and **500** to **-500**.

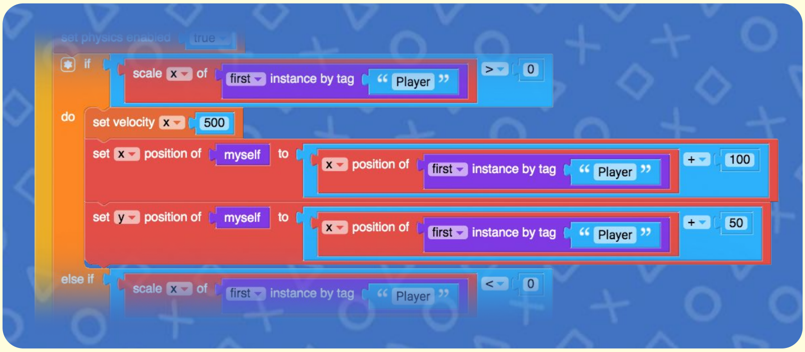


```
When [spacebar] is [pressed]
  set visibility of [myself] to [true]
  set physics enabled [true]
  if  scale [x] of [first] instance by tag "Player"  [>] 0
  do  set velocity [x] 500
  else if  scale [x] of [first] instance by tag "Player"  [<] 0
  do  set velocity [x] -500
```

Close and save the script and hit **PLAY**. When the player presses Spacebar, the projectile should become visible and fire away from the player character.

Next, we will position the projectile in front of the player so that it appears as if they are throwing the object.

Add a **set x position of myself to 0** block to the bottom of the first do. Swap out **0** for a **_+_** block, adding **x position of myself** to the left and **100** to the right. Replace **myself** with **first instance by tag tag name**, and change **tag name** to **Player**.

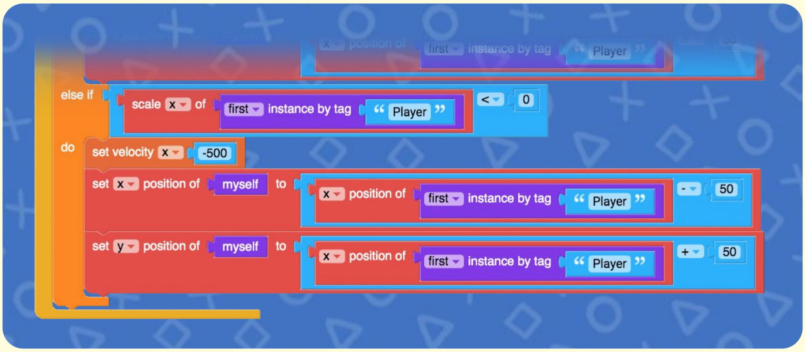


```
set physics enabled  true
if   scale x of  first instance by tag  "Player"  >  0
do   set velocity x  500
     set x position of  myself  to  x position of  first instance by tag  "Player"  +  100
     set y position of  myself  to  x position of  first instance by tag  "Player"  +  50
else if   scale x of  first instance by tag  "Player"  <  0
```

Next, duplicate the red **set x position** block, including everything we just created, and place this new block under the original. Change all **x** values to **y**, and **100** to **50**.

Duplicate both of these blocks and add them to the **do** slot of **else if**. Change the x position of **100** to **50**, and the **+** to a **-**.

Now the player can throw projectiles. Hit the **PLAY** button to test this out.



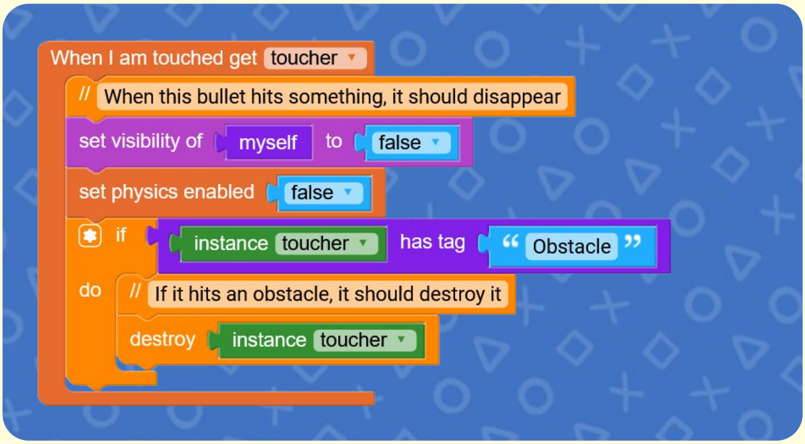
```
     x position of  first instance by tag  "Player"  50
else if   scale x of  first instance by tag  "Player"  <  0
do   set velocity x  -500
     set x position of  myself  to  x position of  first instance by tag  "Player"  -  50
     set y position of  myself  to  x position of  first instance by tag  "Player"  +  50
```

Next, we will code our projectile to detect when it collides with an obstacle, then remove that obstacle from the screen. Open the projectile script.

Drag a **When I am touched get toucher** block into the script. Add **set visibility true** (from Looks) and **set physics enabled true** (from Physics). Change both true values to false.

Underneath these, add an **if do** block. Drag **myself has tag tag name** into it (from Sensing). Replace **myself** with **instance toucher** (from Variables), and change the tag to **Obstacle**.

Finally, grab the **destroy myself** block (from Control Flow), add it to **do**, and replace **myself** with another **instance toucher** block (from Variables).



```
When I am touched get toucher
  // When this bullet hits something, it should disappear
  set visibility of myself to false
  set physics enabled false
  if instance toucher has tag "Obstacle"
  do
    // If it hits an obstacle, it should destroy it
    destroy instance toucher
```

Close the script and save the game. Hit PLAY. Now, the player can throw projectiles and destroy obstacles in their way.

Great work! You've learnt how to use:

☐ If-else control structures

Confirmed by ______________________

SPEED BOOSTS, TELEPORTERS, AND DOUBLE JUMPS

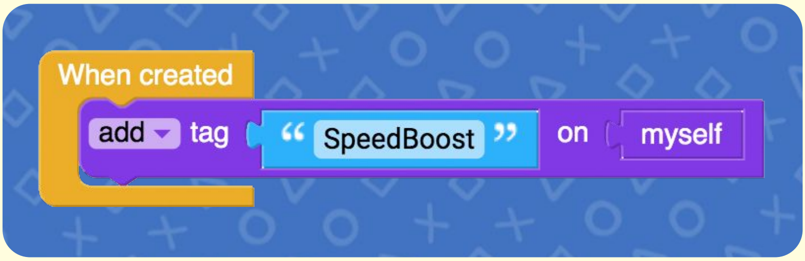
In this chapter, you're going to code player speed boosts and a teleporter. You will also restrict jumping to double jumps so that the player can't stay up in the air indefinitely.

Speed Boosts

Select an object from the Media library to be your speed boost. Place it into the playable level. Right-click on the object and create a script.

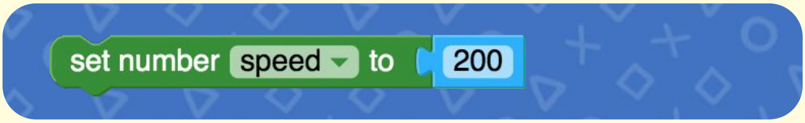
Now tag your speed boost object using a **When created** block (from Event) block and an **add tag tag name on myself** block (from Sensing). Change the tag name to SpeedBoost.

Close and save this script, and name it SpeedBoost and then reopen the player script.



```
When created
    add ▾ tag “ SpeedBoost ” on myself
```

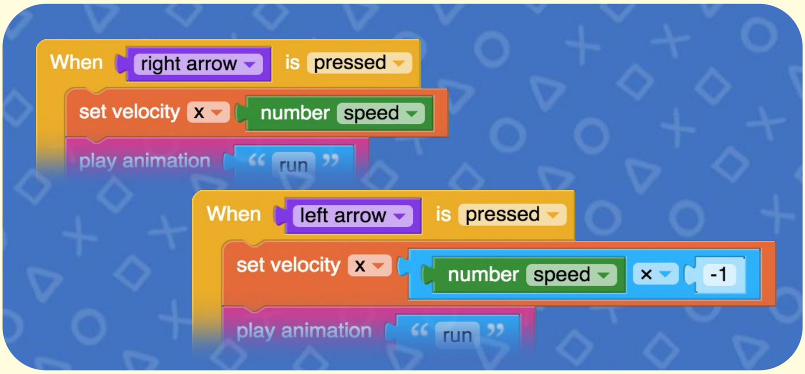
In the player script, find your existing **When level starts** block and create a new number variable using **set true/false i** (Variables). Click on **i** to create a new variable. Name the variable **speed**. Change **true/false** to **number**, and drag a **0** block (from Operators) into the empty space on the right. Change the **0** to **200**.


```
set number speed to 200
```

This variable will allow us to vary the players velocity, which we set up in Chapter 1. Instead of making the player always move at 200 pixels per second, we can update this variable to make the player go faster or slower.

Find your existing **When right arrow pressed** block and replace the **200** block with the **number speed** block (from Variables).

Then do the same with your existing **When left arrow pressed** block. Moving to the left requires negative x velocity, so multiply your speed variable by -1. Drag a **1 + 1** block (from Operators) into your workspace. Place your **number speed** variable inside of the **1 + 1** block on the left. Change **+** to **x**, and change the **1** to **-1**. Multiplying a positive with a negative value always returns a negative number.


```
When right arrow is pressed
    set velocity x number speed
    play animation "run"

When left arrow is pressed
    set velocity x number speed × -1
    play animation "run"
```

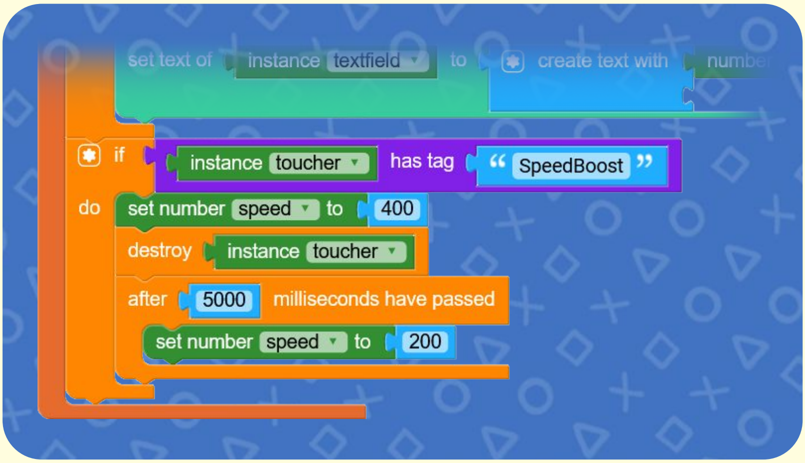
Now, let's make the player detect when it collides with the speed boost object. We will also use a time delay block to set the speed back to normal after a set period of time.

Start by editing the player character script and find the existing **When touched** block. Duplicate one of the **if toucher has tag** sections and change the tag to **SpeedBoost**. Remove the blocks inside the duplicated **if do** block.

Add **set number speed** (from Variables) and set it to **400**. Add **destroy myself** (from Control Flow) and replace **myself** with **instance toucher**.

Add a **100 milliseconds have passed block** (from Control Flow) and change **100** to **5000** (5 seconds).

Duplicate **set number speed to 400** and place it inside **5000 milliseconds have passed**. Change **400** to **200**. This will set speed back to normal.

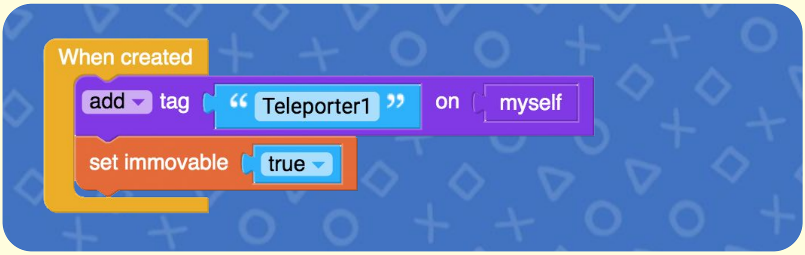


```
set text of instance textfield to create text with numbe
if instance toucher has tag "SpeedBoost"
do  set number speed to 400
    destroy instance toucher
    after 5000 milliseconds have passed
        set number speed to 200
```

Teleporters

Select an object from the Media library to be your Teleporter. Place it into the playable level. Right-click on the object and create a script.

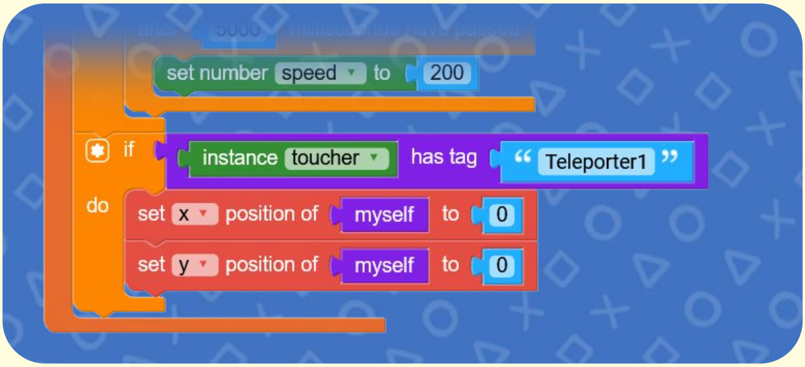
Drag in a **When created** block (from Events) and add a tag using an **add tag** block (from Sensing). Change **tag name** to **Teleporter1** and add a **set immovable true** block (from Physics) underneath it.



```
When created
    add tag "Teleporter1" on myself
    set immovable true
```

Close that script and reopen the player character script. Find the **When I am touched get toucher** block and duplicate one of the **if toucher has tag** blocks. Add this underneath the previous **if do** block and change the tag to **Teleporter1**.

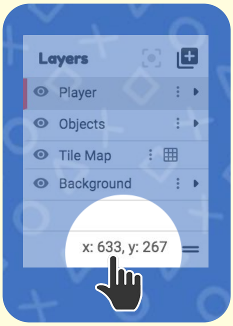
Add a **set x position of myself to 0** (from Transform) inside the **do** slot, duplicate it, and change the duplicated **x** to **y**.



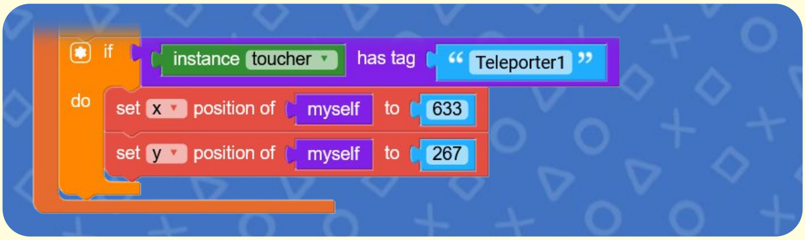
```
        after 5000 milliseconds have passed
            set number speed to 200
    if instance toucher has tag "Teleporter1"
    do
        set x position of myself to 0
        set y position of myself to 0
```

Next we will set the x and y coordinates. When the player touches the teleporter, they will disappear and reappear at these coordinates.

Close and save the player script. In the level editor, use the **Layers panel** to find the x and y coordinates for your target location. Write these down and reopen the player script.



Replace the **0** values from the blocks we just created with the x and y coordinates from the map. Now test your game.



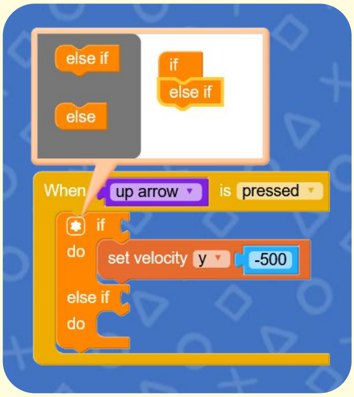
```
if instance toucher has tag "Teleporter1"
do  set x position of myself to 633
    set y position of myself to 267
```

Double Jumps

In this step we will make it so that the player can double jump using if statements with variables.

Open the player script. Find your existing **When up arrow pressed**, and add an **if do** block. Click on its settings icon and add **else if**.

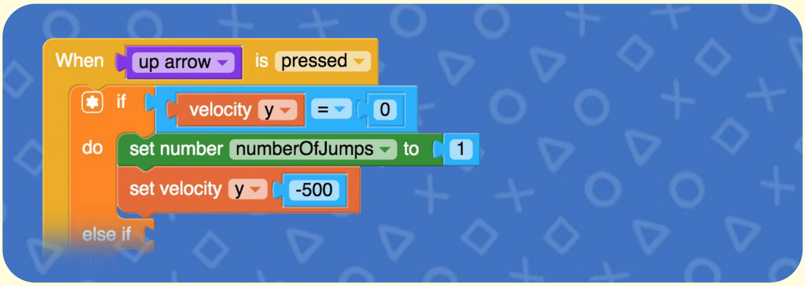
We will use this if block to check if the player is vertically still (not moving up or falling down). Add the **_=_** block (from Operators) to **if**.



```
else if
else
if
else if

When up arrow is pressed
  if
  do  set velocity y -500
  else if
  do
```

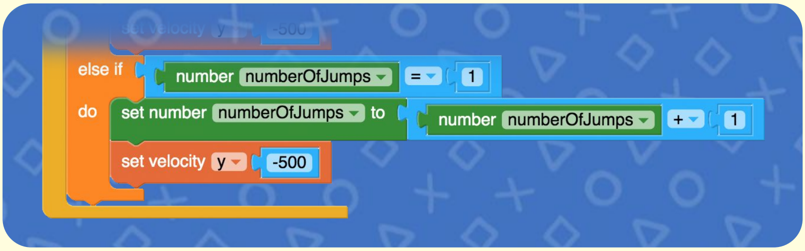
Place **velocity y** (from Transform) on one side of the **=** block, and **0** (from Operators) on the other side. In **do**, create a new number variable called **numberOfJumps** (using Create Variable from Variables) and set it to **1**. Add **set velocity y** (from Physics) underneath with a value of **-500**.



```
When [up arrow] is [pressed]
  if  velocity [y] [=] 0
  do  set number [numberOfJumps] to 1
      set velocity [y] -500
  else if
```

Duplicate **velocity y = 0** and drag it into **else if**. Replace **velocity y** with **number numberOfJumps**, and change **0** to **1**. Then duplicate the two blocks in the first **do** and add them into the second. Replace **1** with **_+_**. Place **number numberOfJumps** on the left and **1** on the right.

Now, play your game to test your speed boosts, teleporters, and double jumps. Great work!



```
      set velocity [y] -500
  else if  number [numberOfJumps] [=] 1
  do  set number [numberOfJumps] to  number [numberOfJumps] [+] 1
      set velocity [y] -500
```

Great work! You've learnt how to use:

☐ User input with selection logic ☐ If-else selection logic

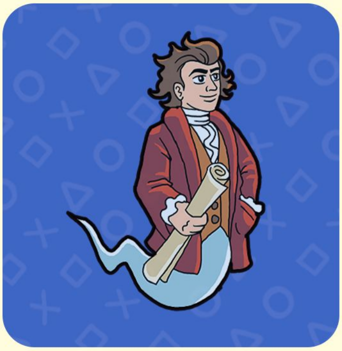
Confirmed by ______________________

CODING A RIDDLE NPC, AND SHARING YOUR GAME

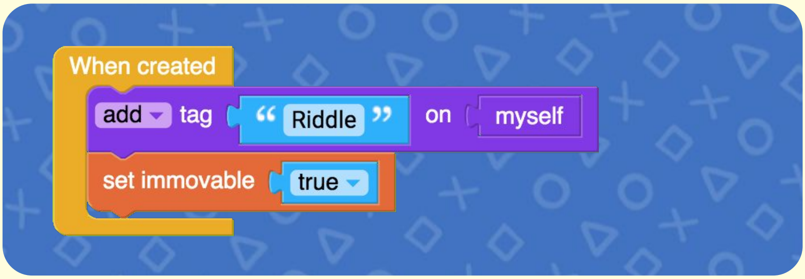
In this chapter, you will add another non-player character to your game, who asks a question or riddle which the player must answer before they can win your game. You will also learn how to publish games so that they can be played by others.

Select a non-player character object from the Media library. Place it into your level. Right-click on the object and click create a script.

Place the NPC in a position so that it is blocking the player from reaching the trophy that will win the game. Add a new script to the NPC object.



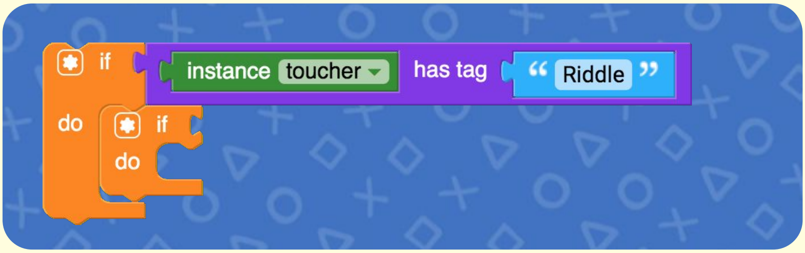
Start with **When created** and give it the tag **Riddle**. Add a **set immovable** to **true** block.



```
When created
    add tag "Riddle" on myself
    set immovable true
```

Now open the player script. Find the existing collision detection code block, and add a new **if do** section in there.

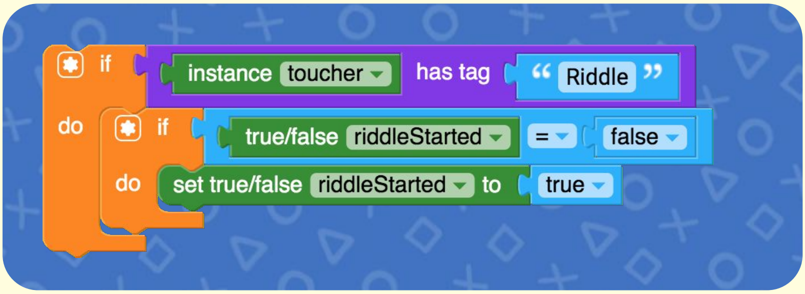
Duplicate an **if instance toucher has tag** block and change the tag to **Riddle**. Drag this into the new **if** section and add another **if do** block inside.



```
if  instance toucher ▾  has tag  " Riddle "
do  if
    do
```

We don't want the player opening the riddle many times at once, so we will check if they haven't yet started the riddle by using a new true/false variable. Grab **true/false i** (from Variables), **_=_** (from Operators), and **false** (from Operators), and connect them in the new if do block. Click on i and create a new variable named **riddleStarted.**

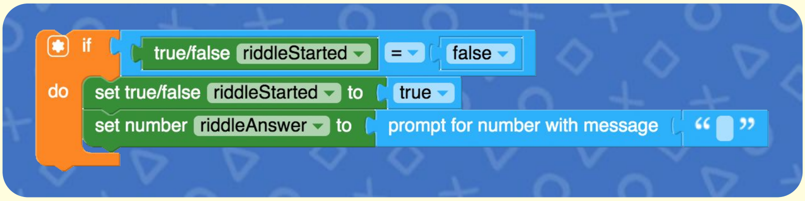
If they haven't started the riddle, **set true/false riddleStarted to true** (from Variables).



```
if  instance toucher ▾  has tag  " Riddle "
do  if  true/false riddleStarted ▾  = ▾  false ▾
    do  set true/false riddleStarted ▾ to  true ▾
```

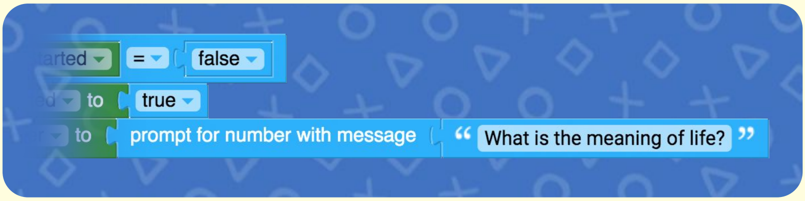
Now we will ask the player a question, and we will save their answer with a new variable. Open Variables and drag **Set true/false i** inside the if do block. Click on i and create a new variable. If you want to ask a riddle with a number as the answer, then set this to a number variable, otherwise set it to a text variable. Name this variable **riddleAnswer.**

To ask the question, grab **prompt for number with message** from Operators (or **prompt for text with message** if you are asking a question with a text answer). Connect the prompt block into your set variable block.



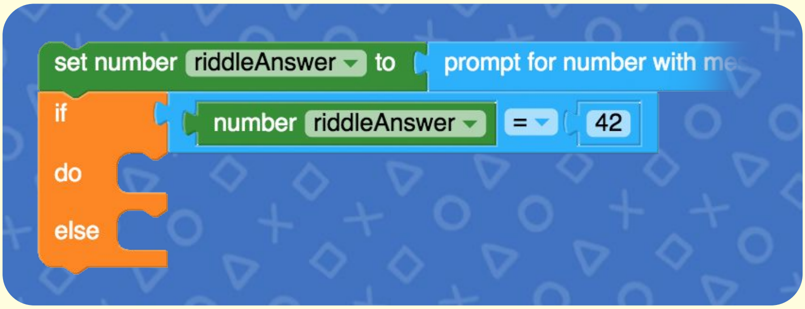
```
if   true/false riddleStarted = false
do   set true/false riddleStarted to true
     set number riddleAnswer to prompt for number with message " "
```

Then type in the question you want to ask inside the quote marks. In our example we are asking "What is the meaning of life?"



```
arted = false
ed to true
   to prompt for number with message "What is the meaning of life?"
```

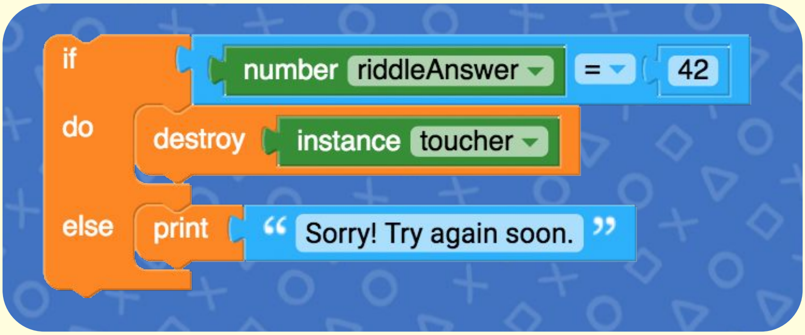
Now we will check if the player typed in the correct answer. Drag an **if do else** block underneath your two variable blocks and attach an **_=_** to **if**. Place **riddleAnswer** in the left side. On the right side we need the correct answer (a number or some text). In this example the correct answer is 42.



```
set number riddleAnswer to prompt for number with me
if   number riddleAnswer = 42
do
else
```

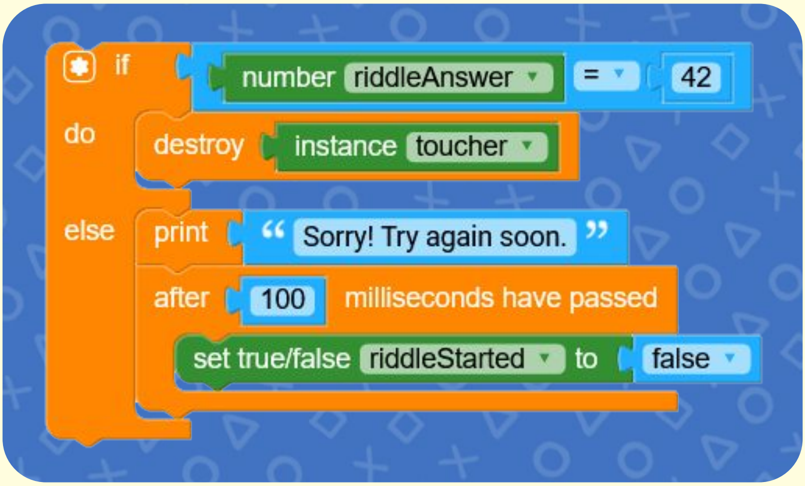
If the answer is correct, we will destroy the riddle object. Inside the **do** block, add a **destroy myself** block. Replace **myself** with **instance toucher** (from Variables).

But what if they got it wrong? Within **else**, add a **print** block and change the text to **Sorry! Try again soon**.



```
if  number riddleAnswer ▾  = ▾  42
do  destroy  instance toucher ▾
else  print  " Sorry! Try again soon. "
```

Lastly, place **after milliseconds have passed** (from Control Flow) underneath the print block. Change the number to **5000** and add **set true/false riddleStarted to false** inside. This resets the riddle to give the player another try.



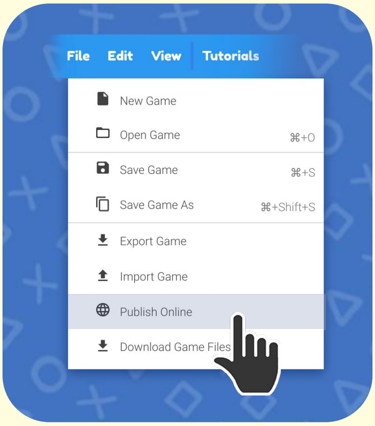
```
if  number riddleAnswer ▾  = ▾  42
do  destroy  instance toucher ▾
else  print  " Sorry! Try again soon. "
      after 100 milliseconds have passed
          set true/false riddleStarted ▾ to false ▾
```

Now, let's publish your game and share it with your friends and family.

To publish your game, **save it**, and then click on **Publish** in the **menu bar**.

From the dropdown menu select **Publish Online**. Give the game a title and hit the **PUBLISH** button.

Gamefroot will compile your game and give you a link. Copy it down and share it with the world.



Great work! You have just built an entire video game. Share it around. Let everyone know that you are a real Video Game Developer!

Great work! You've learnt how to use:

- [ ] User input with selection logic

Confirmed by ______________________

Supported by

GAMEFROOT

Te Puni Kōkiri
MINISTRY OF MĀORI DEVELOPMENT

MINISTRY OF EDUCATION
TE TĀHUHU O TE MĀTAURANGA

PĀTAKA
ART + MUSEUM

With special thanks to

Taiarahia Black, Harko Brown, James Everett,
Jimmy Baird, Luke Smith, Michael Vermeulen,
Malcolm Morrison, Bianca Elkington, Marsella Hippolite,
Reuben Friend, Esme Dawson, Kawika Aipa,

Linda Fordyce, Laura Jones, Lani Evans (and the Vodafone team),
Nick Billowes (and the CORE Education team), Rachel Bolstad,
Heather Moller, Johnson Witehira, Gerard MacManus,
Tim Harford, AKHB, William Young, Benjamin D Richards,
Dave Thornycroft, Dan Milward

and to all of tomorrow's rangatahi!

A games industry partnership between

MINISTRY OF EDUCATION
TE TĀHUHU O TE MĀTAURANGA

GAMEFROOT