TE HIKO TĀKARO

Game Development Club

Make a Platform Game. Learn to Code

GAMEFROOT

This book belongs to:

If found, please return to:

Name:

Email:

Phone:

http://make.gamefroot.com

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

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

Te Puni Kōkiri
MINISTRY OF MĀORI DEVELOPMENT

PĀTAKA
ART + MUSEUM

porirua city

CONTENTS

TE HIKO TĀKARO

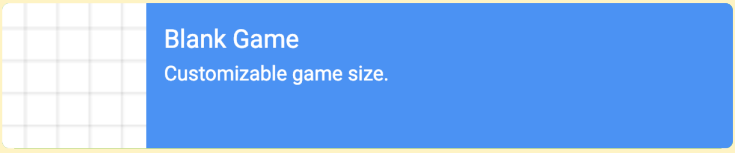
Game Development Club

Introduction: Getting Started in Gamefroot

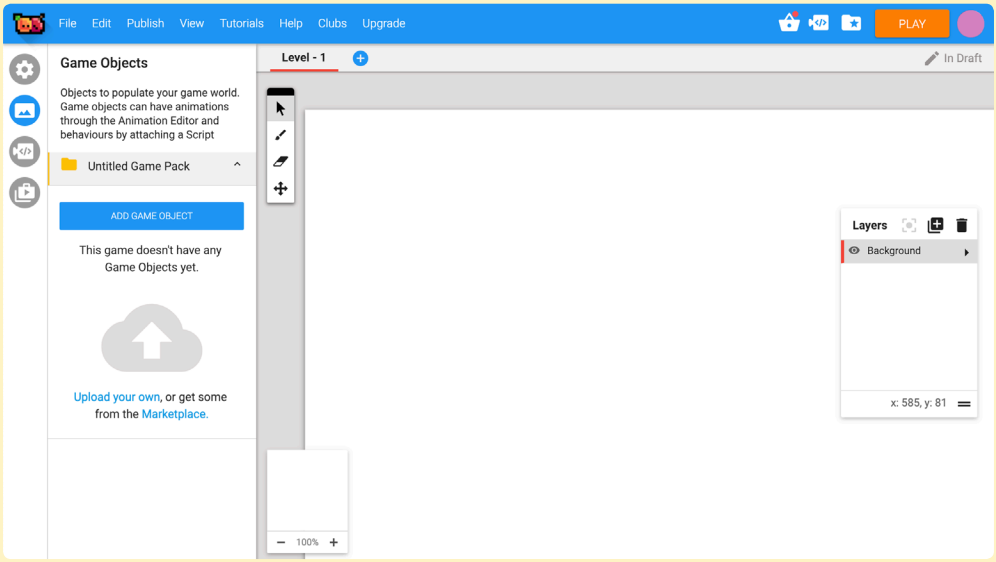
With the help of this Guidebook, we are going to make a platformer game, like Super Mario Bros, and publish it online! Before we get started though, we need to go to the Gamefroot website, open up the Editor, and get everything set up.

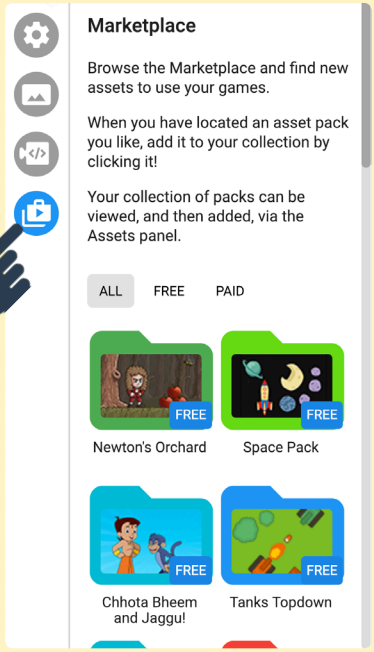
http://make.gamefroot.com

To get started, open your browser and go to the link above. This is the Gamefroot website where you will be making your game. Once on the site, log in or create a new account.



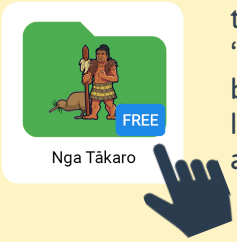
When we sign in, Gamefroot takes us to the Templates page. From here, scroll down to the bottom and select ‘Blank Game’. This opens up the Editor. This contains everything we need to build any 2D game we can think of. Any time you want to create a new game, just go to the Gamefroot website and select ‘Blank Game’.



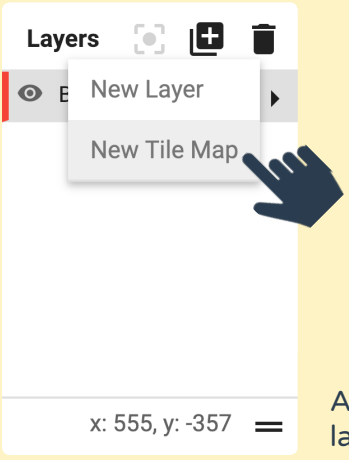


Now that we are in the Editor, we can grab an Asset Pack from the Marketplace. These packs contain collections of characters, objects, artwork and scripts that we can use to build our games.

The Marketplace icon is on the very left of your screen, as seen on the diagram to the left. Click to open the Marketplace, scroll down, 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.

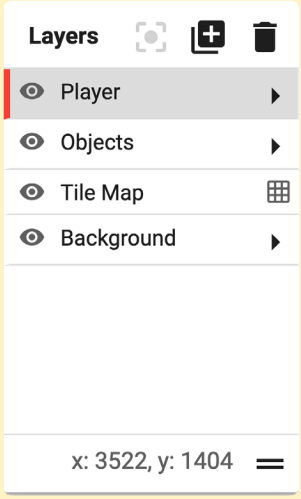


Find the Layers panel. Click on the New Layer icon and select ‘New Tile Map’. Tile assets such as platforms and water tiles can only be drawn on tilemap layers.

After creating the Tilemap layer, create two new layers. The top layer is for the player character. The bottom layer is 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.

You can double-click on a layer to rename it.

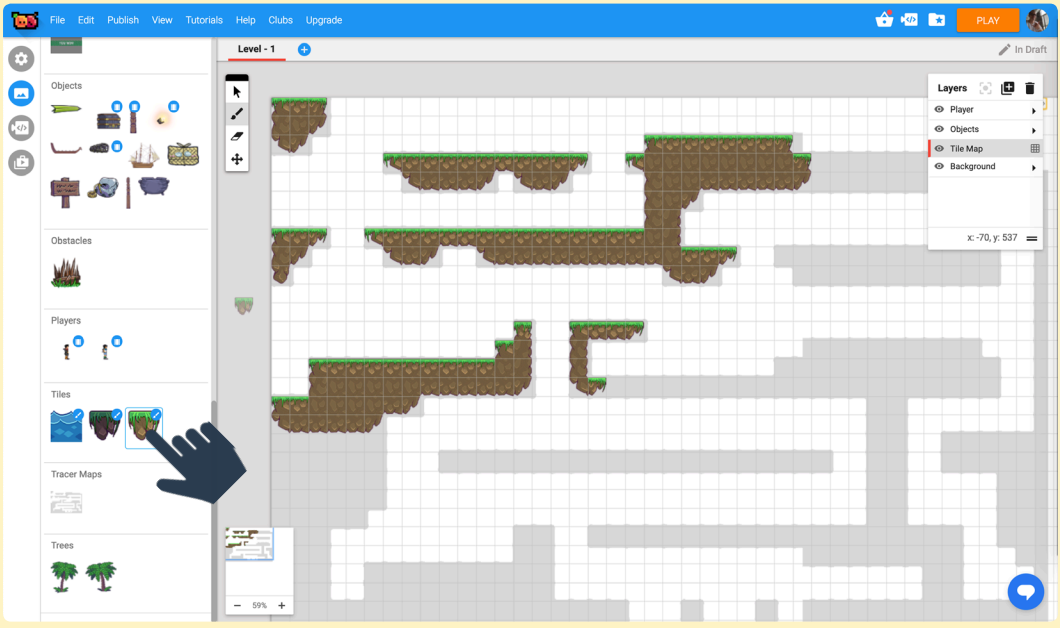


This is the Toolbar, consisting of Select, Brush, Erase, and Pan. Use Select to move, rotate, and scale assets, and use Pan to move your view around the level.

Next, let’s set up a Tracer Map and use it as a guide to help design your first level.

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.

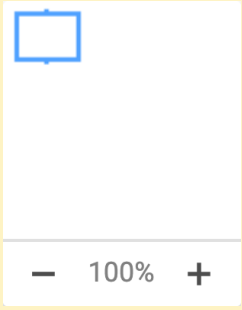
Switch to the Tile Map layer and select the grassy tile brush from the Game Objects panel. Use this brush to create a series of platforms by clicking and dragging across the tilemap.

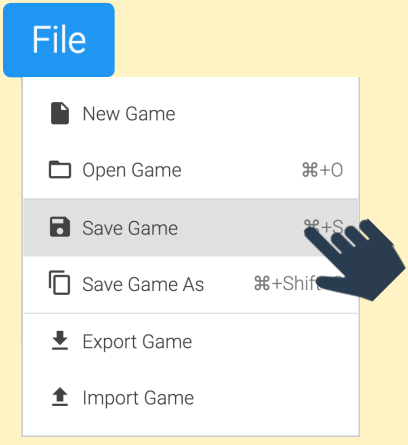


Now, select a playable character for the player to control. Choose either the girl or the boy character.

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

As well as the Move tool, another useful way to move around the level is to use the Minimap. Click on the Minimap to instantly jump to that spot. It's a great way to quickly move around as your levels get bigger.





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

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

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

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:

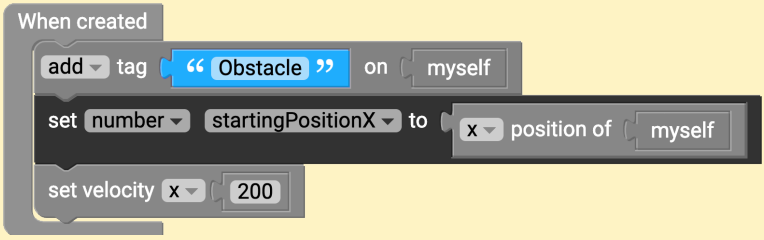
- Improves code readability
- Makes code maintenance easier

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`.



CLASSES and UNIQUE IDENTIFIERS

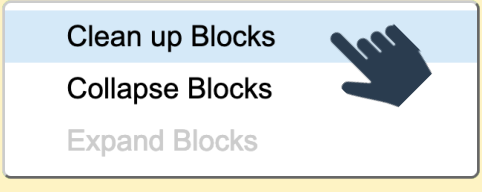
A class is a specific kind of object that can be distinguished from other objects, like a name or type. For example, two obstacles have the same class. Name them in Pascal Case. This is just like Camel Case, but the first letter is capitalised too. For example, `you win` becomes `YouWin`. Use Pascal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

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.



TE HIKO TĀKARO

Game Development Club

Chapter 1: Setting Player Controls with Inputs and Outputs

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

Once we have set up our game, right-click on the player character and click Edit Script. Open the Events panel on the left. Find the `When backspace/delete pressed` block and drag it out into the Script Editor. Change `backspace/delete` to `right arrow`.

Find `set velocity x 0` in Physics and drag it into the `When backspace/delete pressed` block. Change `0` to `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 we are done testing, close the game and return to the Script Editor.

So far, our 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.

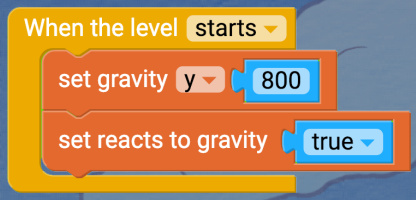
Duplicate the block again and this time, change the new block to `up arrow`. Next, change velocity from `x` to `y` and the number to `-500`.

"Our job as the game creators or developers - the programmers, artists, and whatnot - is that we have to kind of put ourselves in the user's shoes. We try to see what they're seeing, and then make it, and support what we think they might think."

Shigeru Miyamoto

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

To fix this, we will add gravity to the game, then program the player character to stop moving when a direction key is released.

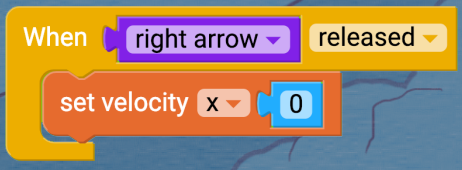


Gravity should start working as soon as the level starts. Grab `When the level starts` from Events. Drag `Set gravity x 0` into it from Physics. Change `x` to `y` and `0` to `800`.

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

I always try and come up with a clear theme when I'm making a videogame.

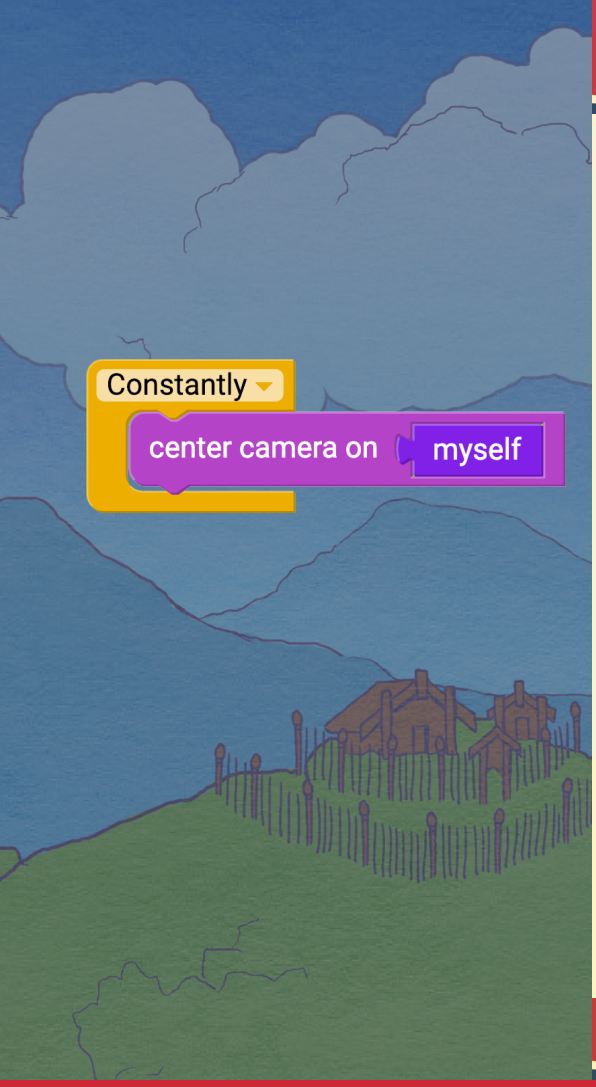
Shigeru Miyamoto



The next step is to detect when a key is released. This is called an Event. We will 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 character move to the right when the right key is pressed.



The final step of Chapter 1 is to set the camera to follow the player. This means that as they move around the level, they won't ever disappear off the side of the screen.

To do this, grab the `Constantly` block from Events. Set the `Center camera on myself` block inside it.

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

If it does, congratulations! You have completed Chapter 1. In the next chapter we will add some deadly obstacles to the game that the player will have to avoid if they want to win.

Great work! In Chapter 1 you've learnt how to use:

- ☐ Inputs & outputs
- ☐ Sequencing
- ☐ Event-driven programming
- ☐ Iteration (loops)

Confirmed by _____

DID YOU KNOW?!

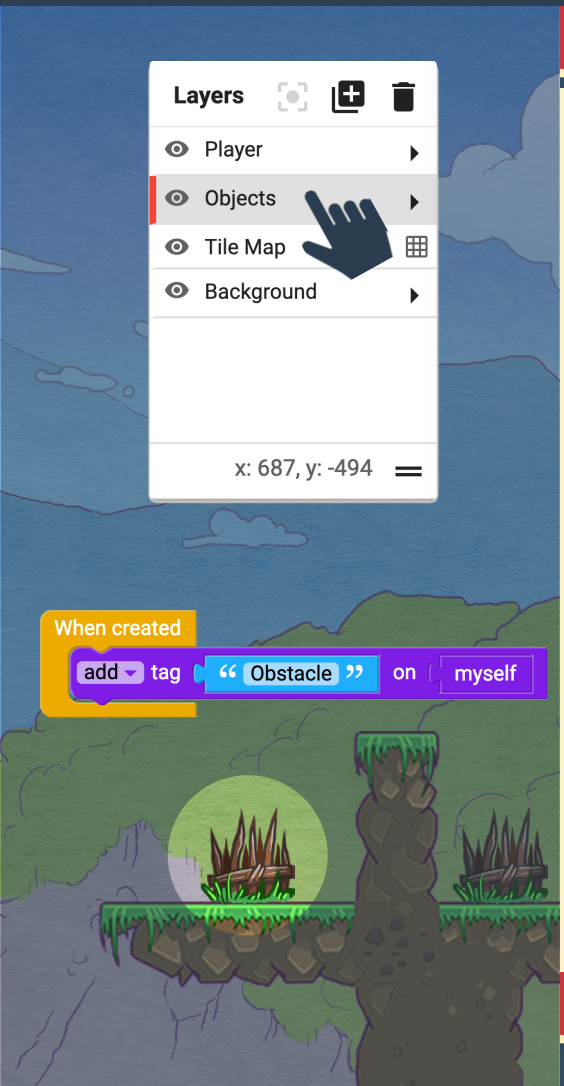
The playable characters in the Nga Tākaro asset pack are called Tama (boy) and Hine (girl).

TE HIKO TĀKARO

Game Development Club

Chapter 2: 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, we're going to edit the player script so that it will detect when it collides with other objects.



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 (see Introduction).

Place some objects onto the level for the player to collide with.

Next, we will add scripts to the objects so that they will be recognised as obstacles.

Right-click on one of the obstacles and click Add Script. Drag out the code block `When created` and place `Add tag tag name on myself` inside it.

Change `tag name` to `Obstacle` and close the Script Editor. Save the script as `Obstacle`.

Now, add this script to all of our obstacles. We can reuse this script on any object in our level that we want to be collidable.

To do this, open the Scripts sidebar, 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.

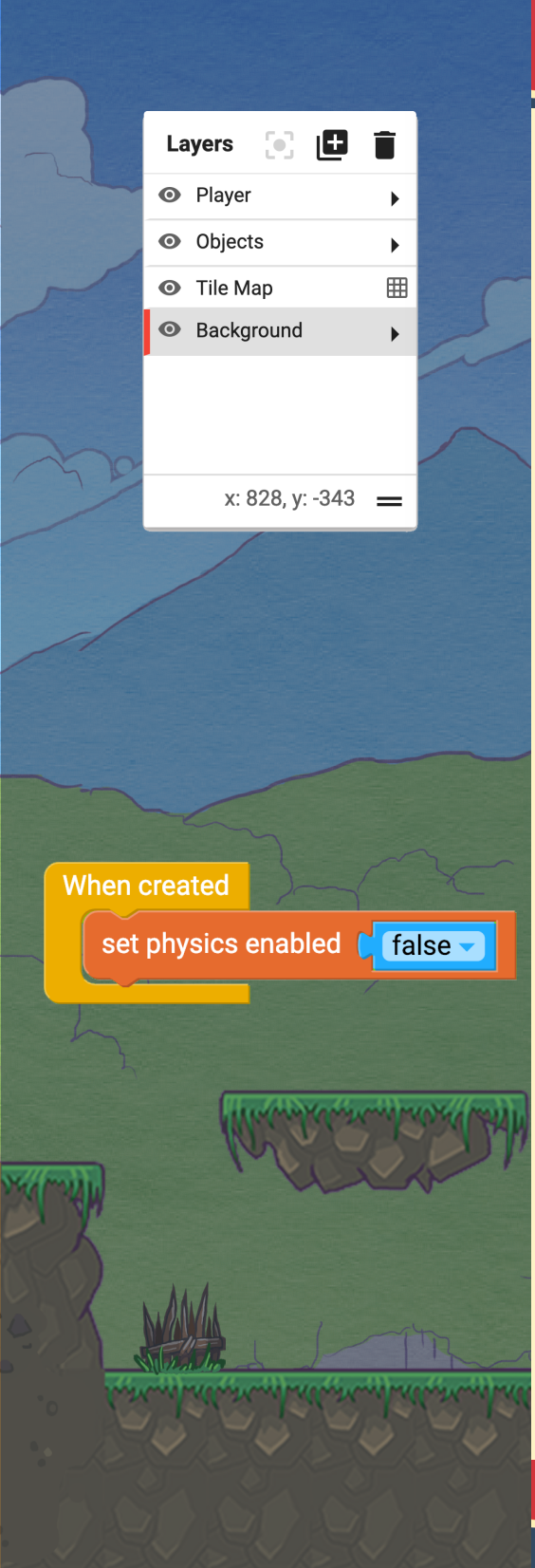
Next, add code to the player so that the level restarts when it collides with an object.

Right-click on the player character and click Edit Script. Add the following code blocks; `When touched get toucher`, `if do`, `myself has tag tag name`, and `instance toucher` from Variables.

Replace `myself` with `instance toucher`. Change `tag name` to `Obstacle`.

Put `instance toucher has tag Obstacle` in the first `if` slot. Put the `if do` block in `When I am touched get toucher`.

Finally, find the `go to next level` block and add it to `do`. Change `next` to `current`.



Close the script and save it. Hit PLAY to test what we have built so far. The game should restart every time the player collides with an obstacle.

> "Games have grown and developed from this limited in-the-box experience to something that's everywhere. Interactive content is all around us."
>
> Shigeru Miyamoto

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. Place your background image onto this layer and right-click on it to add a new script.

We will tell the game not to apply collision rules to the background image.

Find the `When created` block and drag `set physics enabled true` into it. Change `true` to `false`.

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

Drag the `Constantly` block out into the Script Editor. Find `set x position of myself to 0` and set it inside `Constantly`. Replace `0` with the `camera x` block. Duplicate `set x position of myself to camera x` and change both `x` values to `y`.

Play the game. If the background now follows the player, congratulations! You have completed Chapter 2.

Great work! In Chapter 2 you've learnt how to use:

- ☐ **Selections with conditional logic**
- ☐ **If statements**

Confirmed by _____

DID YOU KNOW?!

Kupe discovered New Zealand around AD 950. Kupe's wife, Kuramārōtini, suggested the new land be called Aotearoa, the Land of the Long White Cloud.

TE HIKO TĀKARO

Game Development Club

Chapter 3: Level Navigation and Game States

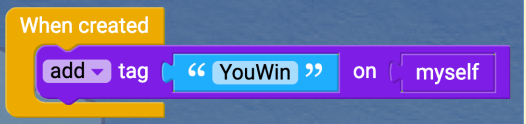
In Gamefroot, game levels are like chapters in a book. Players work their way through one, and when finished, move on to the next. In this chapter, you will add levels to your game and learn how to navigate between them.

Add a new level to your game. Click the blue and white plus button next to `Level 1`. Double-click on the level's name to rename it. Rename the level to `You win`. Do the same again, but name this new level `You lose`.

Add objects and decorations to each of these levels to make them into real You win or You lose screens.

By the end of this chapter, you will have set up the following levels:

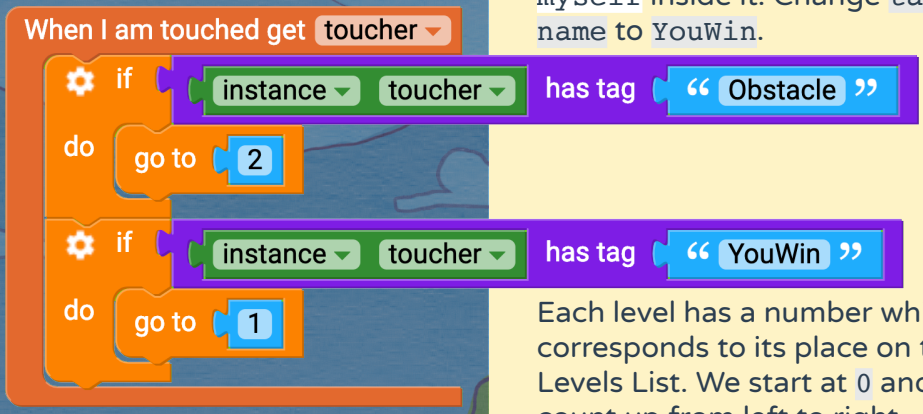
- Start screen
- Your playable level
- You win screen
- You lose screen



Next, we will add a trophy to the game. When the player touches it, they will be shown our new `You win` screen.

To add this trophy, switch back to the playable level and place a new object to be the trophy.

Add a new script to this object using a `When created` block. Place `add tag tag name on myself` inside it. Change `tag name` to `YouWin`.

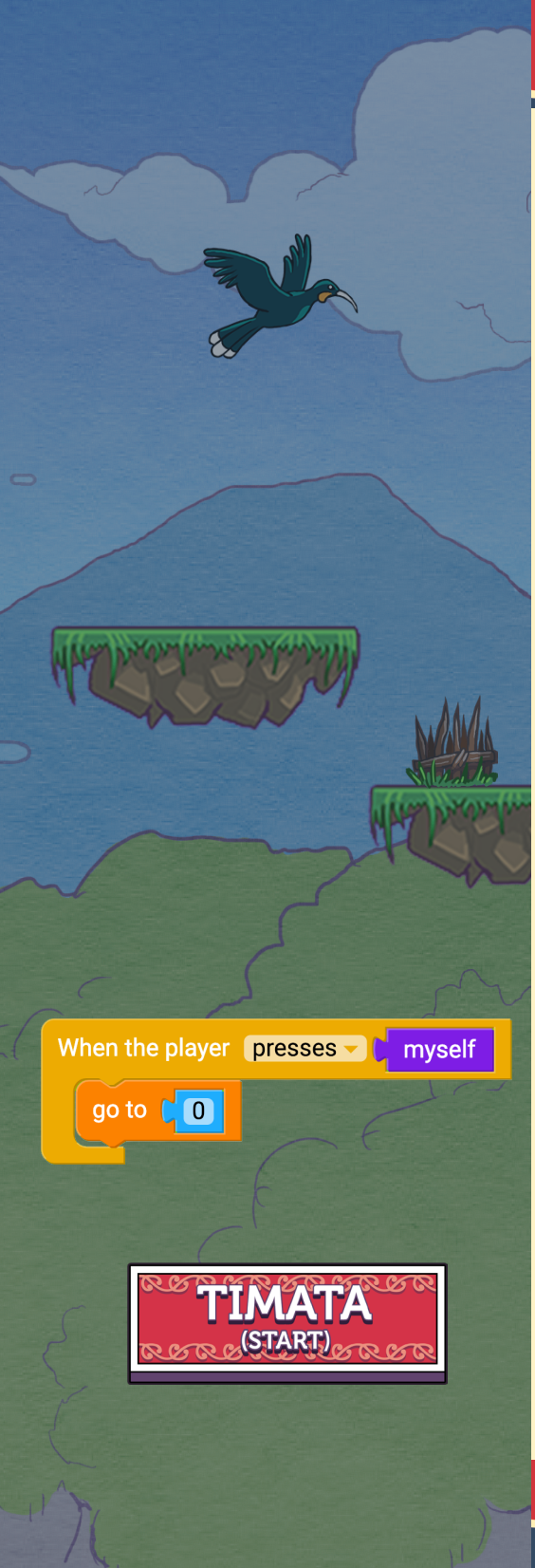


Each level has a number which corresponds to its place on the Levels List. We start at `0` and count up from left to right.

Now, make the player detect when it collides with the trophy. Go back to the player character's script and add another `if do` block to our collision detection code.

"The objective of video games is to entertain people by surprising them with new experiences."

Shigeru Miyamoto



Connect a `myself has tag tag name` block to the `if` and change the tag to `YouWin`. Add `go to next level` to `do`. Replace `next level` with a `0` from Operators. Change `0` to the level number for `You win` (1).

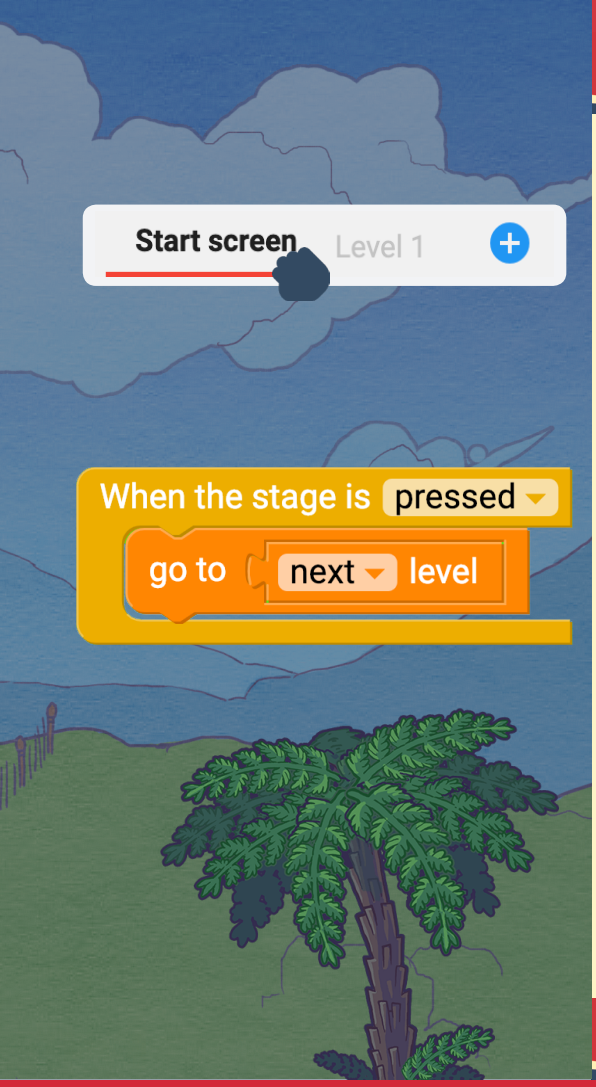
We will now change the collision detection code to send the player to the `You lose` screen instead of resetting the level.

Edit the player character's script and change `go to current level` to another `go to` block. Change the number to the game's `You lose` level number.

In the next step, we will program in a Replay button. When the player loses, they can hit this button to try again.

Make sure that both the You win and You lose levels have a Timata button to start over. Add a new script to one of these buttons which includes `When the player presses myself` with `Go to next level` inside it. Change `next` to `first` and save the script.

Open the Scripts panel and add this new script to the other Timata button.



Add a new level and call it `Start screen`. Drag that level to the front of the list and place objects in it to create the start screen.

As we moved this level to the start, increase the level numbers from the last page by +1.

Include a Click to Start button. The player will click on this to start the game. Add a script to Click to Start with `When stage is pressed`, and `go to next level` inside it.

Great work on Chapter 3! 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! In Chapter 3 you've learnt how to use:

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

Confirmed by _____

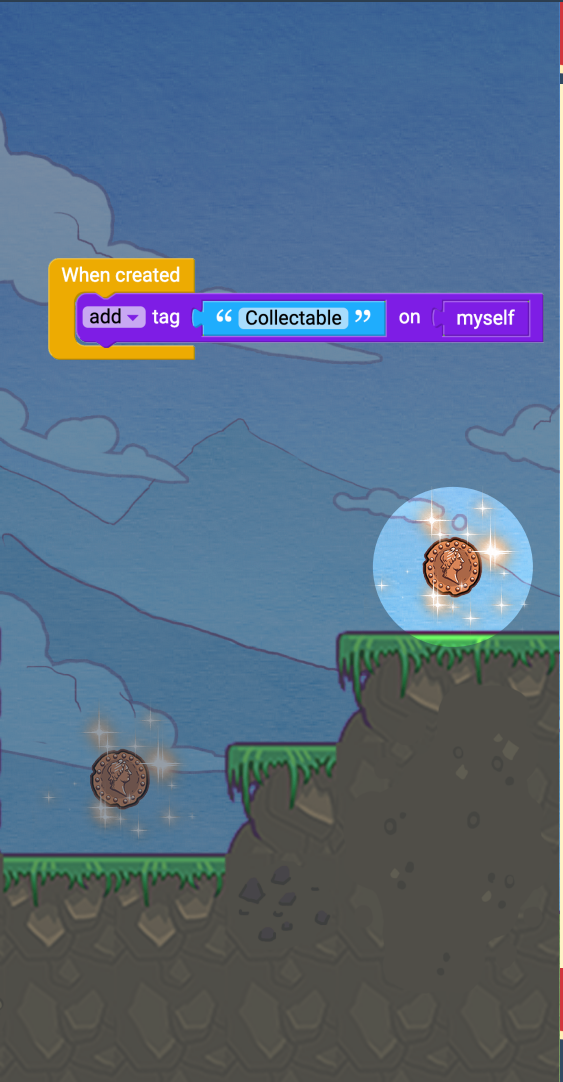
DID YOU KNOW?! **Matariki is the name of a cluster of stars which rise in mid-winter and signal the start of the Māori new year.**

TE HIKO TĀKARO

Game Development Club

Chapter 4: Keeping Score with Variables

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



Start adding objects to the playable level to act as collectable items.

Once you have placed your collectable items, add a new script to one of them. Add in the code block `When created` with `add tag tag name on myself` inside it. Change `tag name` to `Collectable`.

Close and save the script. Open the Scripts panel and add the new script to all of your collectables.

Now that we have collectables set up, we are going to create a number variable to act as a scoring system. This will increase every time the player touches one of the collectables.

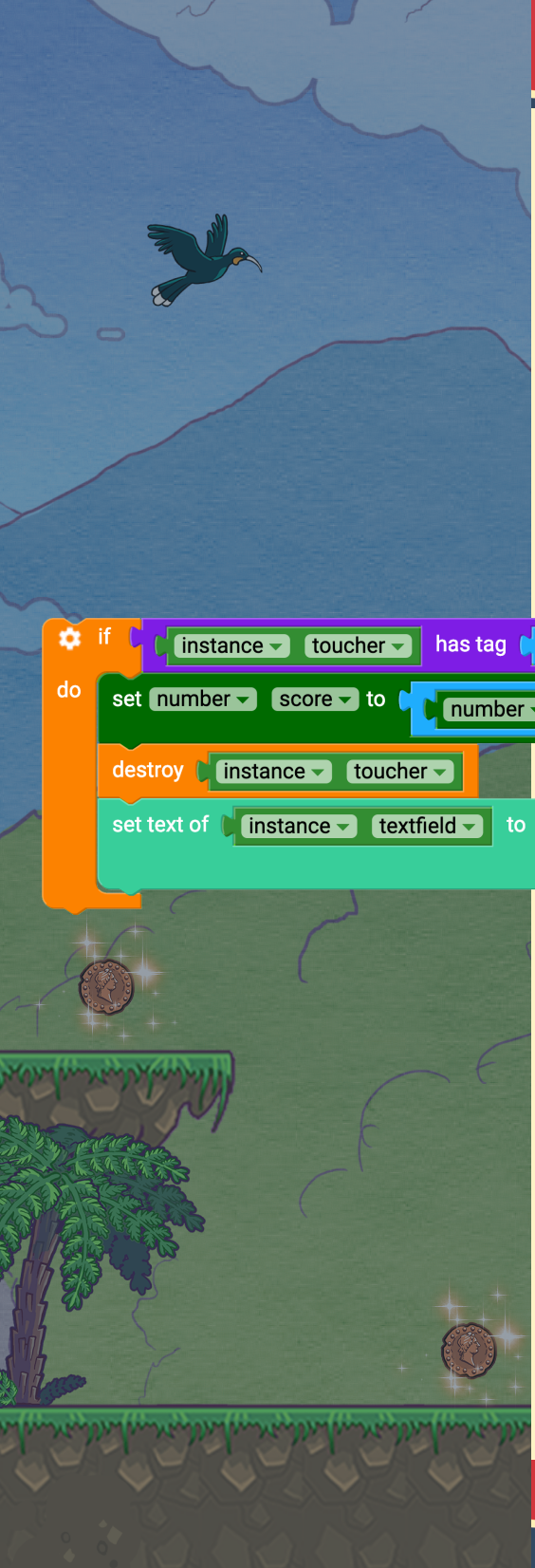
Edit the player character script. Open the Variables panel. Add `set true/false i` to the existing `When the level starts` block, at the bottom.

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

Next, grab `create new textfield` from Draw and connect it to the bottom. Type `0` into the textfield.

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

Close and save the script. Hit the PLAY button to test out the new scoring system.



Notice that the score number doesn't increase when you collect one of the collectables.

To get the score textfield to update, add `Set text of instance textfield` from the Draw panel to our collectable collision detection block, under `destroy instance toucher`.

Now, grab the `number score` block and put it in `set text of instance textfield to`. But it won't connect! This is because it is a number value. The `set text of` block expects text, not a number.

To resolve this, convert it from a number 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.

Nicely done! You have completed Chapter 4. In the next chapter, we're going to learn how to code a patrolling obstacle that the player must avoid.

Great work! In Chapter 4 you've learnt how to use:

- ☐ **Variables**
- ☐ **Data types**

Confirmed by _____

DID YOU KNOW?!

Māori in the 1800s traded timber, pork and potatoes for muskets and rum. Dressed flax (harakeke) was the most sought-after item by foreign traders.

TE HIKO TĀKARO

Game Development Club

Chapter 5: Coding a Moving Obstacle

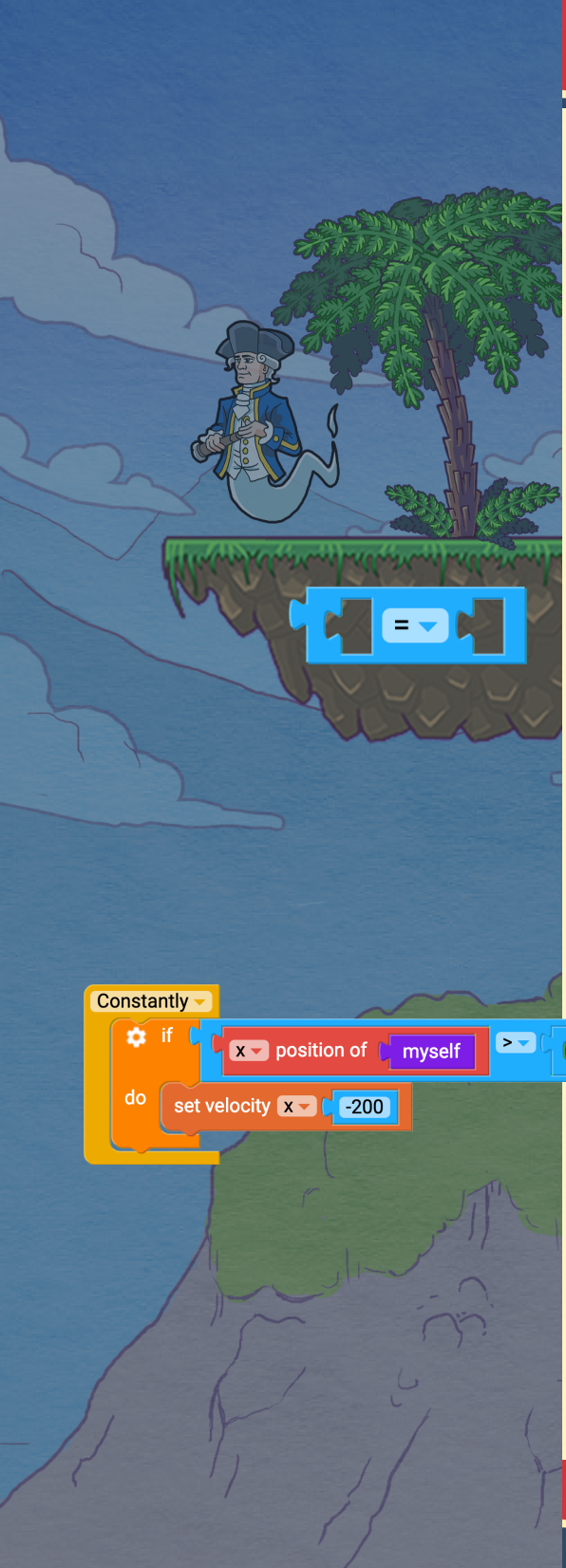
In this chapter, we will program an obstacle that patrols back and forth from one place to another. We will save the initial position as a variable, start the obstacle moving in a direction, and then check how far it's moved from its original position.

Add an obstacle to be our Patroller, making sure it has enough space to move around.

Add a new script to this object and start with the `When created` block. Drag `Add tag` into it and change `tag name` to `Obstacle`.

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

Set `x position of myself` to the end of `set number starting PositionX`. Add `set x velocity` below, and change `x` to `200` (see image).



In this next step, we will stop the patrolling object wandering too far away from where it started. We do this by comparing its current position to its original position, using a comparator.

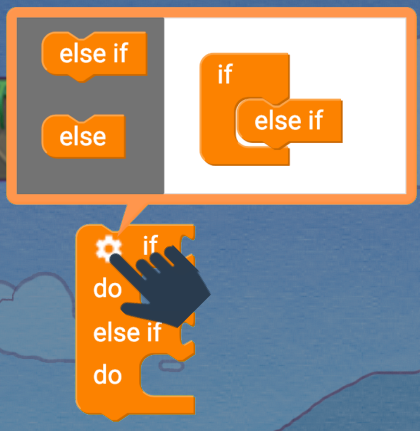
Grab the `Constantly` block and drag an `if do` block into it.

Add the `_=_` comparator to the if slot. This is in the Operators panel, in the Boolean section (see image). Change it to `_>_` (greater than). Add `x position of myself` to the empty space on the left, and `_+_` to the right of the `_>_`.

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.

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` statement block we made. Change it to an `if else if` statement. This is used for programming more complex decision making.

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).

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.

Next, we will make the player character flip to face the correct direction as they move around the level. This is done by changing the object's scale and centering its anchor point.

Reopen your player character's script and drag `set scale x of myself to 0` into the `When left arrow pressed` block. Change `0` to `-1`. Duplicate this block and add it to `When right arrow pressed`. Change `-1` to `1`.

Finally, find `center anchor point of myself` and add it to `When the level starts`.

Ka pai! You've completed Chapter 5 and learnt how to code a deadly moving obstacle.

Great work! In Chapter 5 you've learnt how to use:

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

Confirmed by _____

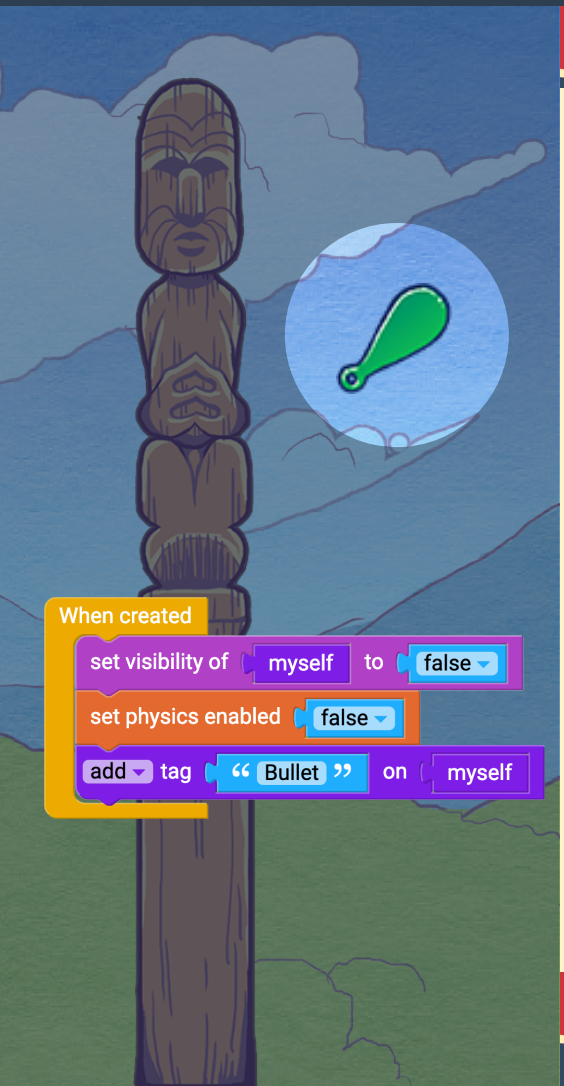
DID YOU KNOW?! **Captain James Cook's first visit to New Zealand was in 1769, his second in 1773, and his third in 1777.**

TE HIKO TĀKARO

Game Development Club

Chapter 6: Adding and Throwing Projectiles

In this chapter, we will create an object that our player will be able to throw while in the game. We 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 and place `set visibility true`, `set physics enabled true`, and `add tag tag name on myself` inside it. Change both `true` values to `false`. Change `tag name` to `Bullet`.

Close and save the script.

Open up the player script. Find the existing `When the level starts` block, and add `add tag tag name on myself` to it. Change `tag name` to `Player`.

Next, we will make a keyboard trigger which makes the player character throw the projectile.

In this case, we will use the Spacebar as our trigger. Open the projectile script and add `When backspace/delete pressed`. Change `backspace/delete` to `spacebar`.

Drag `set visibility true` and `set physics enabled true` into `when spacebar pressed`. Change both `false` values to `true`.

> "I think what's really the most ideal thing is for the player themselves, within their own imagination, to carve out what they view as being the essence of the character."
>
> Shigeru Miyamoto

Grab an `if do` block and add it into `when spacebar pressed`. Click on its settings icon and change it to an `if else if` block.

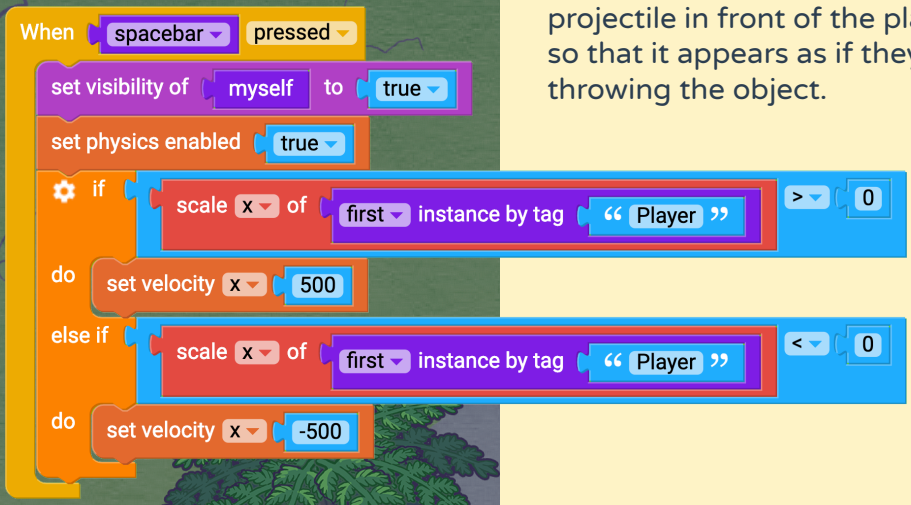
Add a `_>_` (greater than) block to the first `if` slot. Place `scale x of myself` to the left and `0` to the right. Replace `myself` with `first instance by tag tag name`, and change `tag name` to `Player`.

Add a `set velocity x` block to the first `do`, and set it to `500`.

Duplicate both of these blocks and add them to the `else if` section. Change `>` of `_>_` to `<`, and `500` to `-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.

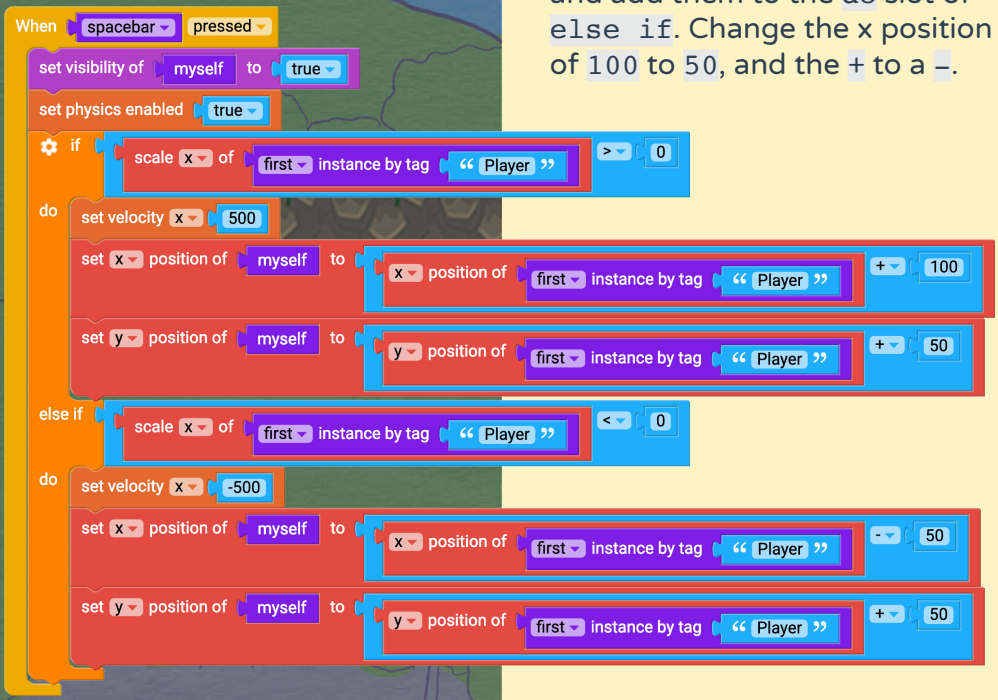
Next, we will position the projectile in front of the player so that it appears as if they are throwing the object.



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`.

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.

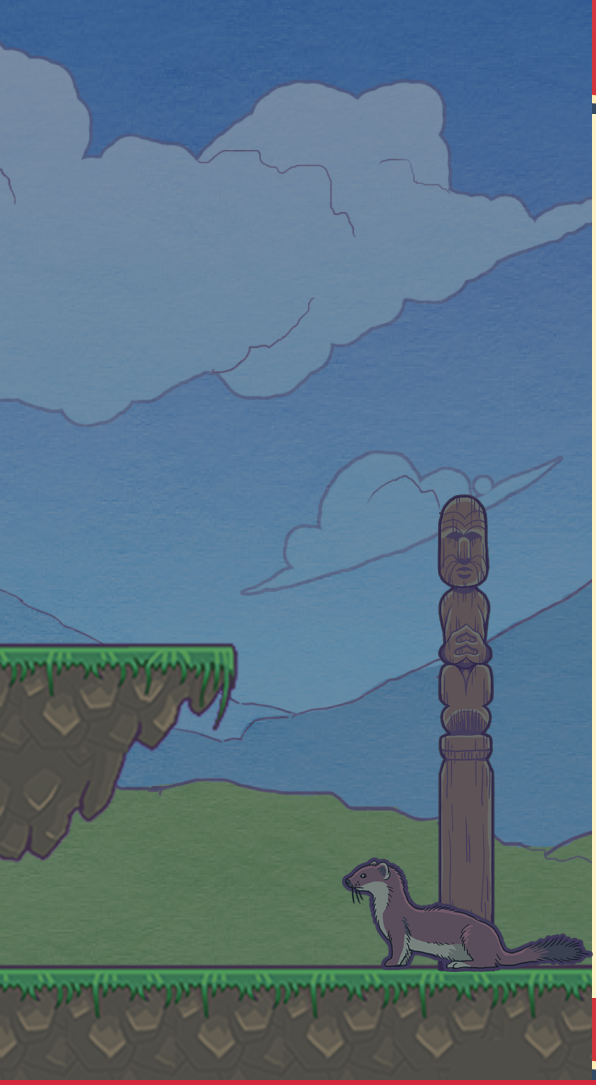
Next, we will code our projectile to detect when it collides with an object, then remove that object from the screen.

Open the projectile script.

Drag a `When I am touched get toucher` block into the script. Add new `set visibility true` and `set physics enabled true` blocks to it. Change both `true` values to `false`.

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

Finally, grab `destroy myself`, add it to `do`, and replace `myself` with another `instance toucher` block.



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

Good job, and great work on completing Chapter 6! You have learnt how to code throwing projectiles. In the next chapter, we will learn how to create speed boosts and teleporters, and add double-jumping to the player!

"When I'm making video games today, I want people to be entertained. I am always thinking, How are people going to enjoy playing the games we are making today?"

Shigeru Miyamoto

Great work! In Chapter 6 you've learnt how to use:

- ☐ **If-else control structures**

Confirmed by _____

DID YOU KNOW?!

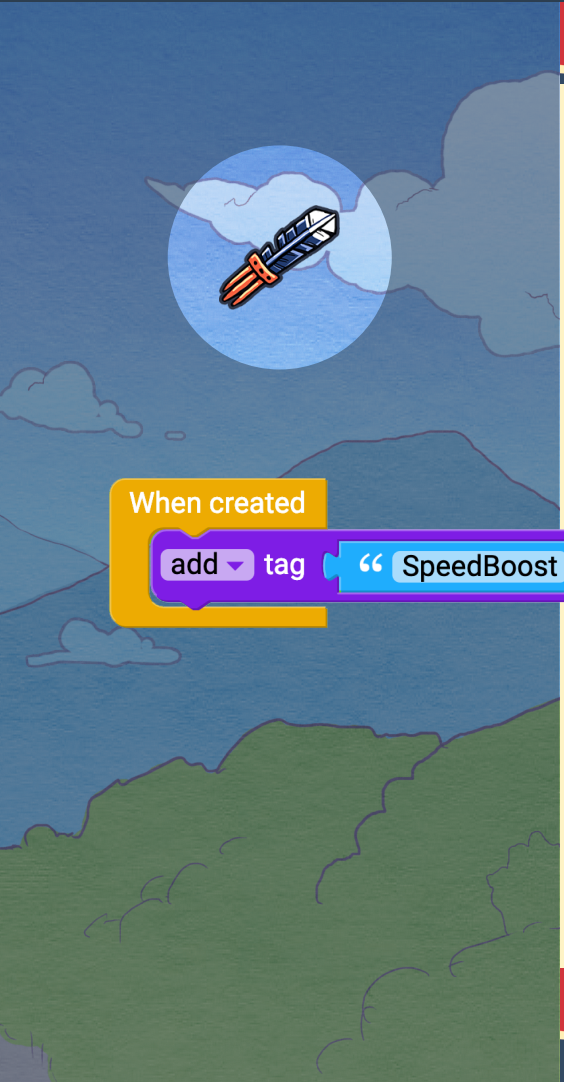
Traditional Māori weapons were made of wood, stone, and bone. Most were designed for hand-to-hand combat.

TE HIKO TĀKARO

Game Development Club

Chapter 7: Speed Boosts, Teleporters, and Double Jumps

In this chapter, we will add a few more advanced mechanics to the game. We will create and program speed boost collectible objects and a teleporter. We will also restrict jumping to double jumps so that the player can't stay up in the air indefinitely.



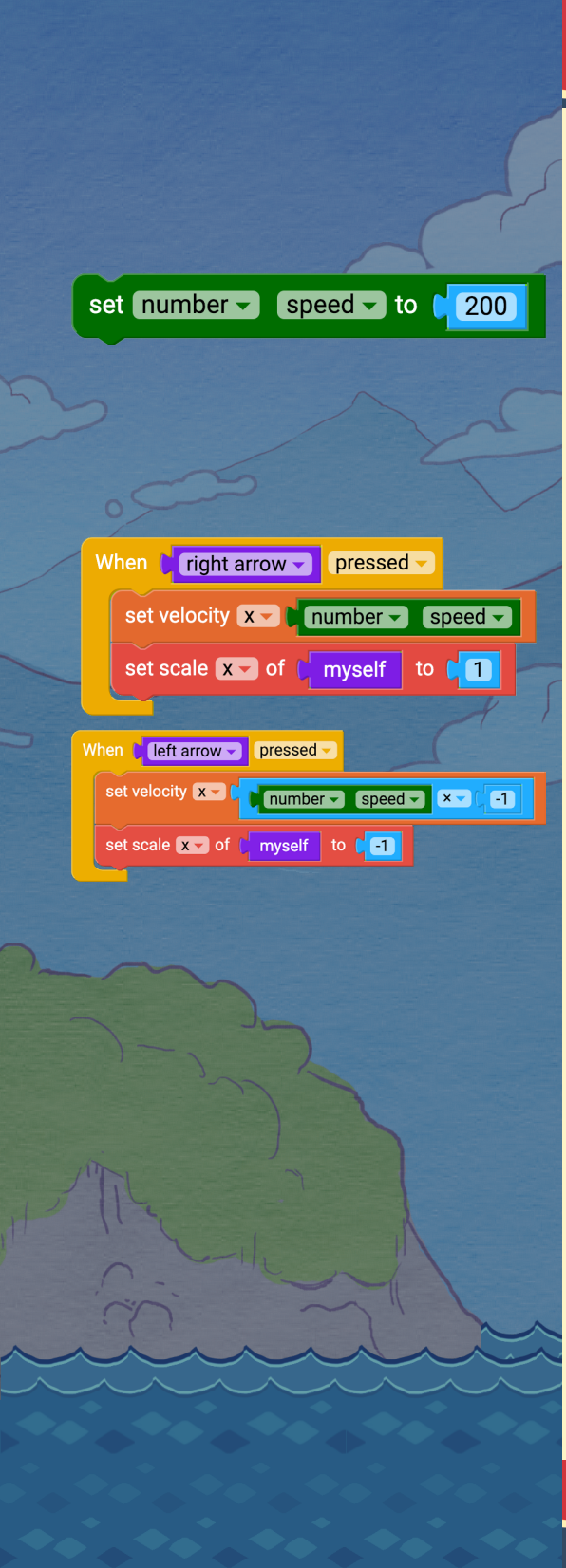
SPEED BOOSTS

First, we will set up the speed boost object and a corresponding speed variable on the player.

Start by selecting your speed boost object. Place it into the playable level. Right-click on the object and create a script.

Add a tag to the object using `When created` and `add tag tag name on myself`, changing `tag name` to `SpeedBoost`.

Close and save this script, calling it `SpeedBoost`, and reopen the script on the player character.



Navigate to the existing `When level starts` block and create a new number variable using `set true/false i to`. Click on `i` to create a new variable called `speed`. Change `true/false` to `number`, and drag a `0` block into the empty space on the right. Change the `0` to `200`.

This sets the default speed value to 200. We then need to go back and change our existing directional movement blocks to use this variable, instead of the set values we used in Chapter 1.

To do this, find `When right arrow pressed` and replace the `200` block with the variable `number speed`. Do the same with the `When left arrow pressed` block, but note that we are replacing a negative number. Place `number speed` inside of a `_+_` block. Change `+` to `x`, and multiply the variable by `-1`. This will take our speed variable, whatever it is, and multiply it by -1 to give us a negative value.

In the next step, we will add code to increase the speed, and then reset it back to its original value once the speed boost wears off.

Now, we will 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`.

Add `set number speed to` and set it to `400`. Add `destroy myself` and replace `myself` wth `instance toucher`.

Add a `100 milliseconds have passed` block, 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.

“I always try and come up with a clear theme when I’m making a videogame.”

Shigeru Miyamoto

TELEPORTERS

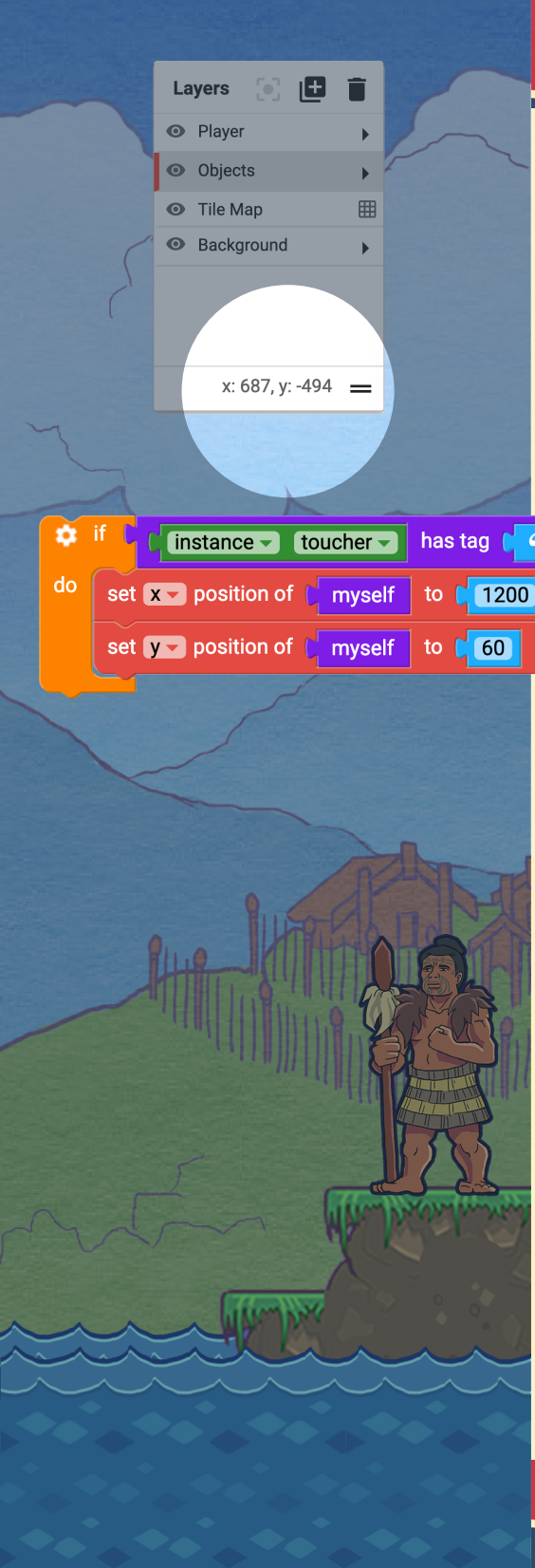
Start by placing an object onto the playable level. We will use this as the teleporter. Add a new script to it.

Drag in a `When created` block and add a tag using an `add tag` block. Change `tag name` to `Teleporter1` and add `set immovable true` underneath it. Close the script and reopen the player character script.

Go to our `When I am touched get toucher` block and duplicate one of the `if toucher has tag` blocks. Add this to the bottom and change the tag to `Teleporter1`.

Add a `set x position of myself to 0` to the `do` slot, duplicate it, and change `x` to `y`.

Next, we have to set a pair of x and y coordinates. When the player touches the teleporter, they will disappear and reappear at the selected spot.



Close the script. In the level editor use the Layers panel to find the x and y coordinates for a point on the map. 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 our point on the map.

DOUBLE JUMPS

In this last step, we will make it so that the player can only jump once, then once again while in the air. We will use `if` statements with variables to create our double jump script.

Open up the player script.

Find `When up arrow pressed` and add an `if do` block. Click the `if do` settings icon and add an `else if`.

Add `_=_` to `if`. Place `velocity y` on one side and `0` on the other.

In `do`, create a new number variable called `numberOfJumps` and set it to `1`. Add `set velocity y` underneath with a value of `-500`.

Duplicate `velocity y = 0` and drag it into `else if`. Replace `velocity y` with `number numberOfJumps`, and change `0` to `1`.

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!

In the next chapter we will wrap everything up, publish our game, and share it with others.

Great work! In Chapter 7 you've learnt how to use:

- ☐ **If-else selection logic**
- ☐ **Variables**

Confirmed by _____

DID YOU KNOW?! **The group of 12 feathers from a huia's tail was called a mareko, and was worn by high chiefs going into battle.**

TE HIKO TĀKARO

Game Development Club

Chapter 8: Finishing and Sharing Your Game

In this final chapter, we will add a character to our game who asks a riddle which the player must answer before they can win. We will also learn how to publish games so that they can be shared with, and played by, others.

Place an object into the level. We will code it to ask the player a riddle. They must answer this riddle correctly before they can win the game.

Place the Riddle object 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 Riddle object.

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

Open the player script and find the exisiting collision detection code block.

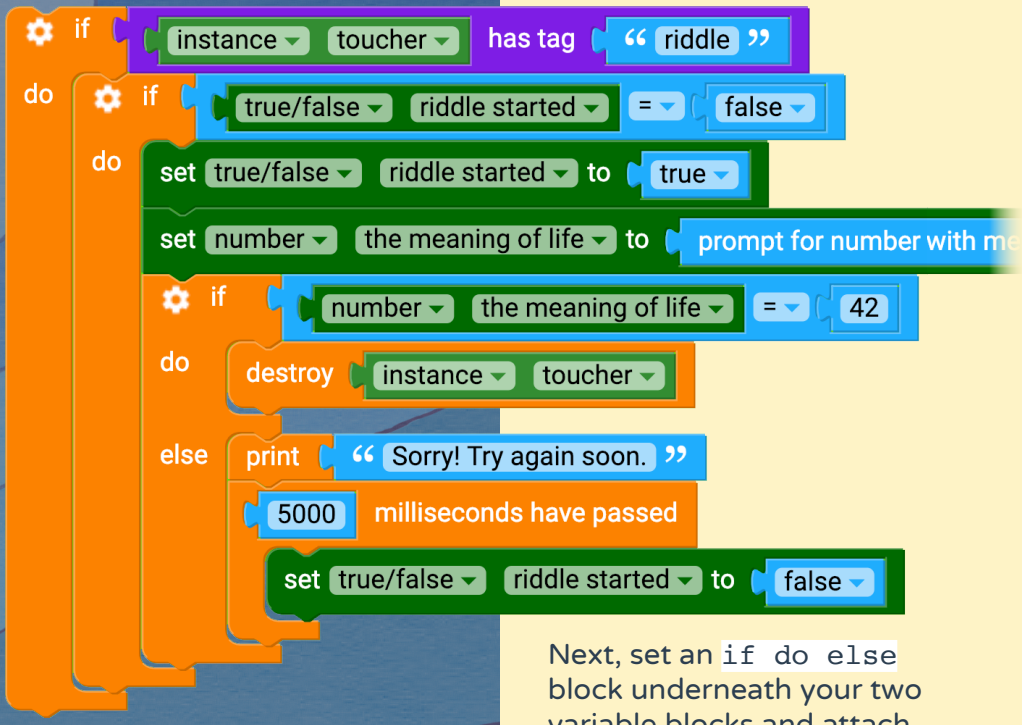
Add a new `if do` section here.

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.

In the new `if` section, add a variable to check if the player has already answered the riddle correctly. Do this with an `_=_` block. Set `true/false i` to the left and `false` to the right. Click on `i` and create the variable `riddleStarted`.

Add `set true/false riddleStarted to`, and set it to `true`.

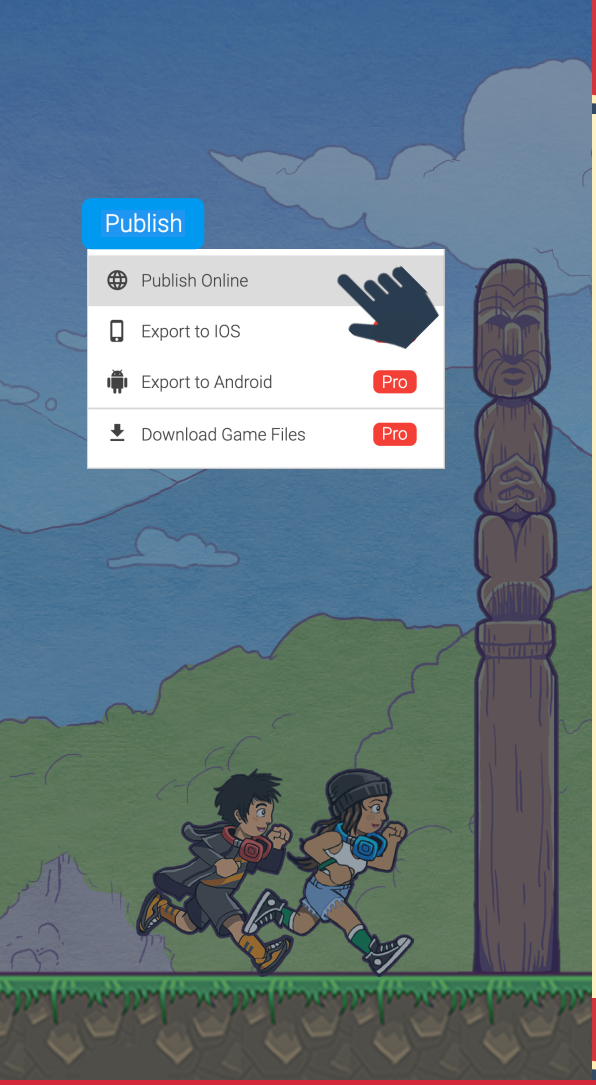
Below this, add another variable. This time, set it to a new number variable called `theMeaningOfLife`. Add `prompt for number with message` from Operators. Set the message to `What is the meaning of life?` (or whatever question you want to ask).



Next, set an `if do else` block underneath your two variable blocks and attach an `_=_` to `if`. Add `number theMeaningOfLife` to the left side and the number `42` to the right.

Within the `do` block, add a `destroy myself` block. Replace `myself` with `instance toucher`.

Finally, within `else`, add a `print` block and change the text to `Sorry! Try again soon`. Set a `milliseconds have passed` block underneath. Change the number to `5000` and add `set true/false riddleStarted to false` inside.



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! In Chapter 8 you've learnt how to use:

- ☐ **User input with selection logic**

Confirmed by _____

DID YOU KNOW?!

Kaitiakitanga means guardianship and protection. How could you use games to be a kaitiaki and preserve Aotearoa's unique history and stories?

Supported By

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, Laura Scatchard, Craig Briggs,
AKHB, William Young, Benjamin D Richards,
Dave Thornycroft, Christopher Elson, Dan Milward

and to all of tomorrow's rangatahi!