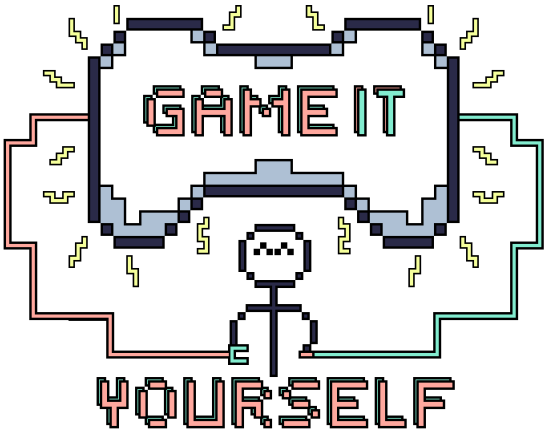

Game-It-Yourself Toolbox

Level 2: Poll of Lies

Did you know that a 2018 American study showed that girls who play first-person shooters score, on average, 15 points higher in IQ than boys who also play these games?

That and many other unsupported and unfounded claims circulate the internet daily. In a world filled with quick, bite-sized content like TikToks, YouTube Shorts, Instagram posts and Reels, X posts, *etc.*, we frequently accept information at face value, simply due to the lack of time and energy to fact-check everything. Sometimes people share personal beliefs and generalized opinions as facts or genuine advice, making it hard for the information to not sound trustworthy. Other times, people with malicious intentions start sharing chains of misinformation just to provoke a reaction and observe how far it spreads. Even mainstream media sources fail at times, wreaking havoc and promoting fear by stating half-truths or information that hasn't been verified by research or unbiased third parties. This can lead to unnecessary panic in situations where such reactions may not be justified.

It is, however, our duty as internet users to ensure we double-check with multiple sources and cross-analyse opposing opinions. This is crucial to staying educated and informed, especially when it comes to information regarding healthcare! To avoid adopting harmful habits or making the wrong choices, we must be extra careful about information regarding our bodies and how they function, what is safe and healthy and what is not. Ultimately, remember to always seek advice from a medical professional if and whenever possible.

universidade de aveiro
DigiMedia digital media and interaction research centre
FCT Fundação para a Ciência e a Tecnologia
COMPETE 2020
PORTUGAL 2020
UNIÃO EUROPEIA Fundo Social Europeu

This activity seeks to promote awareness and prowess in healthcare media literacy by playing, developing and designing a 2D digital game. This program was created in the context of the research project **PLAYMUTATION2** (UIDB/05460/2020) and is directed towards adolescents (12-18 years old) and young adults (19-35 years old) to promote the development of **STEAM** competencies: Science, Technology, Engineering, Arts and Mathematics.

Poll of Lies is a 2D game that challenges the player to **think critically** about the information contained in different social media posts, trying to figure out which ones are true based on **media literacy principles**, such as checking the source and its reputation, verifying the date, recognizing bias, among others. The development of this game is carried out using **Godot**, a **beginner-friendly game engine** that does not require installation. Godot uses a simple, high-level, programming language, GDScript, and has updated documentation and active forums to discuss information.

Level of Skill: Beginner and Intermediate.
Time to Complete: Expect to spend 1 to 2 hours on this mission, depending on your current level of skill.

Skills covered in this course

Concepts:

→ Familiarization with media literacy principles;

→ Promote critical thinking;

→ Disseminate factual healthcare information through player choices.

Game Development:

→ Familiarization with the Godot 4.1+ game engine;

→ Implement button click actions;

→ Control scene changes;

→ Create and add new content to the game;

→ Display score.

Programming:

→ Interpret code;

→ Use logic structures to control the flow of execution of code (selection, i.e. if-else conditions);

→ Understand data types, variables and data structures;

→ Familiarize with GDScript syntax;

→ Write easy-to-read and efficient code that integrates into an existing system;

→ Diagnose and fix bugs, compilation errors, exceptions and code that does not perform as expected;

→ Comprehend and apply object-oriented programming principles.

Plan of Activities

- **Mission 1:** Media Literacy is NOT (just) for Nerds

Quickly learn the basic concepts and principles of media literacy through a matching pairs game and understand how to efficiently conduct a Google search.

- **Mission 2:** GO DO (I)T!

Download Godot and import a project that you will later help develop.

- **Mission 3:** Explain Godot to Me Right Now or I Quit

The game isn't working?! Roll up your sleeves, we are fixing this!

- **Mission 4:** But I Hate Programming

Game development is a multidisciplinary field with space for everyone! If you are this generation's Van Gogh, try creating an animated character for the game.

Contents

1 Media Literacy is NOT (just) for Nerds.

1.1 Mission description

To be able to navigate through the complex media landscape of today's digital age, especially with the surge of artificial intelligence, we must think critically about the information we are shown - and we are shown A LOT of it, more so than ever before. But fear not! There are a few **principles of media literacy** that you can easily implement into your life to help you better understand which sources are trustworthy, up to date and with minimal bias.

After that, to put some of that new knowledge into practice and learn **how to efficiently make a Google search**. Some of these tips can be applied to other search engines, such as YouTube, DuckDuckGo, Yahoo, Bing, *etc.*

1.2 Goals

1.2.1 Media Literacy Made Easy

To complete this mission you must successfully undertake a challenge: a game of matching pairs. Connect each media literacy principle to its correct definition, aiming for the highest score in the smallest amount of time. See if you can reach the top position in the leaderboard!

Ready? To play, click here.

1.2.2 Let Me Search That For You: How to Google?

To optimize your Google searches and go directly to the point, there are some simple tips and tricks that you can implement. Let's learn them through a simple example: (see Figure 1).

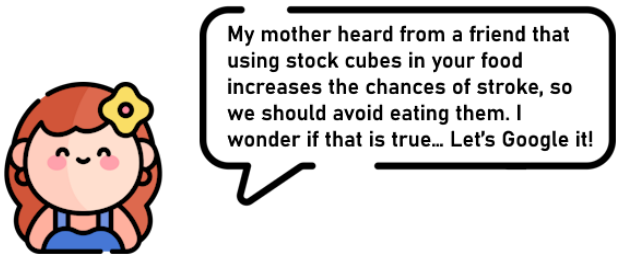

Figure 1: Example: My mother heard from a friend that using stock cubes in your food increases the chances of stroke, so we should avoid eating them. I wonder if that is true... Let's Google it!

1. Think about the question or topic you want to search about and identify the main keywords, these are the most important aspects of the question. Be specific to narrow the results closer to what you actually need (see Figure 2).

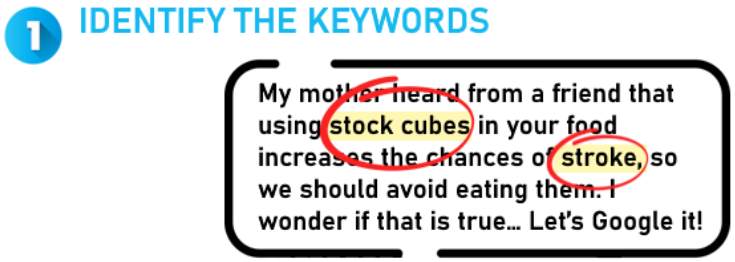

Figure 2: Highlight the keywords.

2. If you are specifically looking for a certain word or one of the keywords you thought of is an expression composed of more than one word, you can prompt Google to show you results that include an **exact match** to what you have searched. To do this, simply wrap said expression between quotation marks, "like so"! (See Figure 3).

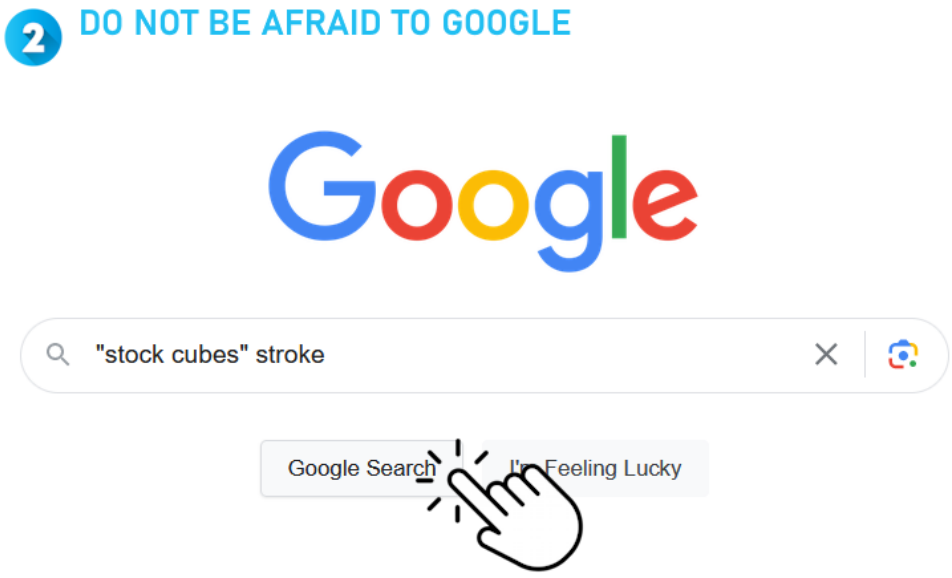

Figure 3: Quotation marks for an exact match.

3. To search for information on a specific website, place site: followed by the website of your choice and the keywords you are looking for, for example, site:wikipedia “stock cubes” (see Figure 4).

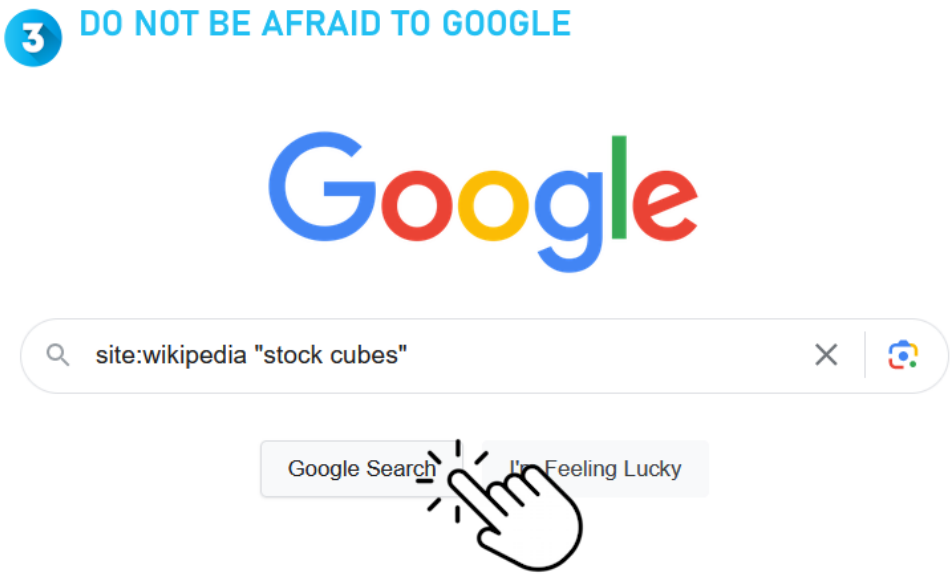

Figure 4: Search in specific site.

4. Say you do not want results that contain a certain term or expression. To do this, place a minus sign, - , behind that expression, like this: how to make stock -cubes . (See Figure 5).

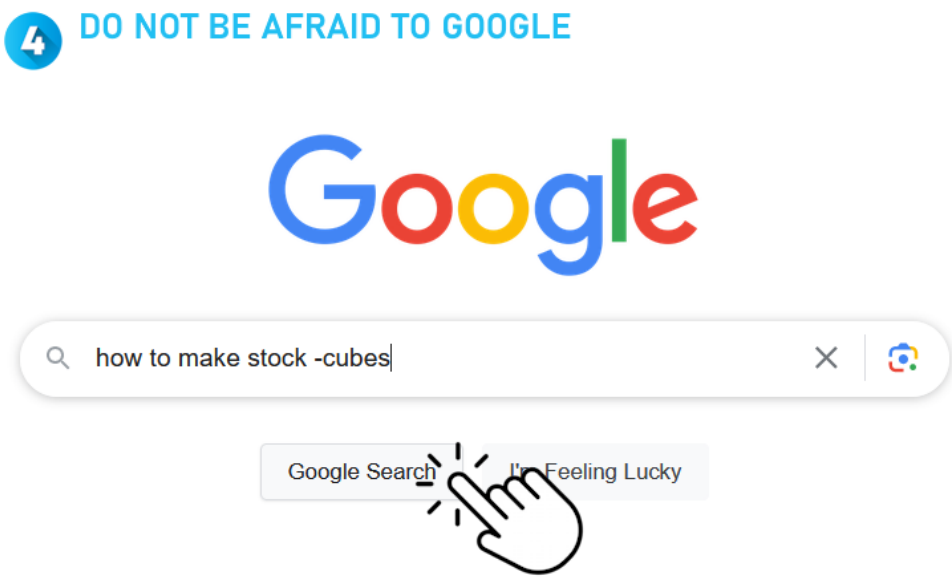

Figure 5: Minus sign to exclude results with a certain expression.

5. Your results will not always show up at the surface level. Sometimes you must go deeper and scroll away until you find what you need (see Figure 6).

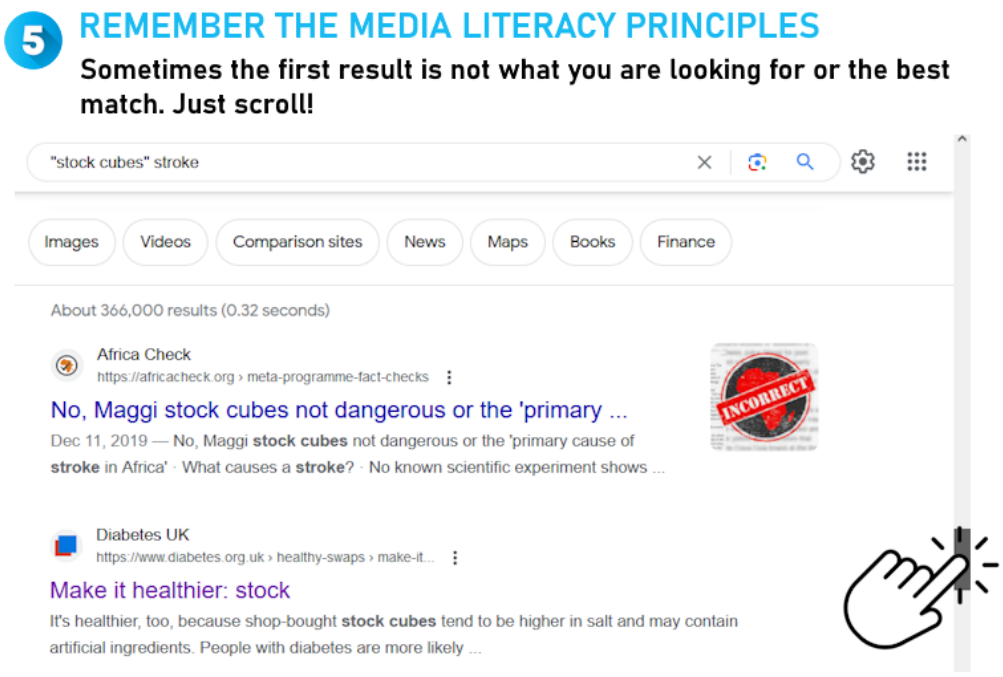

Figure 6: Scroll, scroll, scroll!

6. Click the three dots next to a search result to know more about the source and why Google's systems determined this would be a useful result. (see Figure 7 and 8).

Figure 7: Three dots next to a search result.

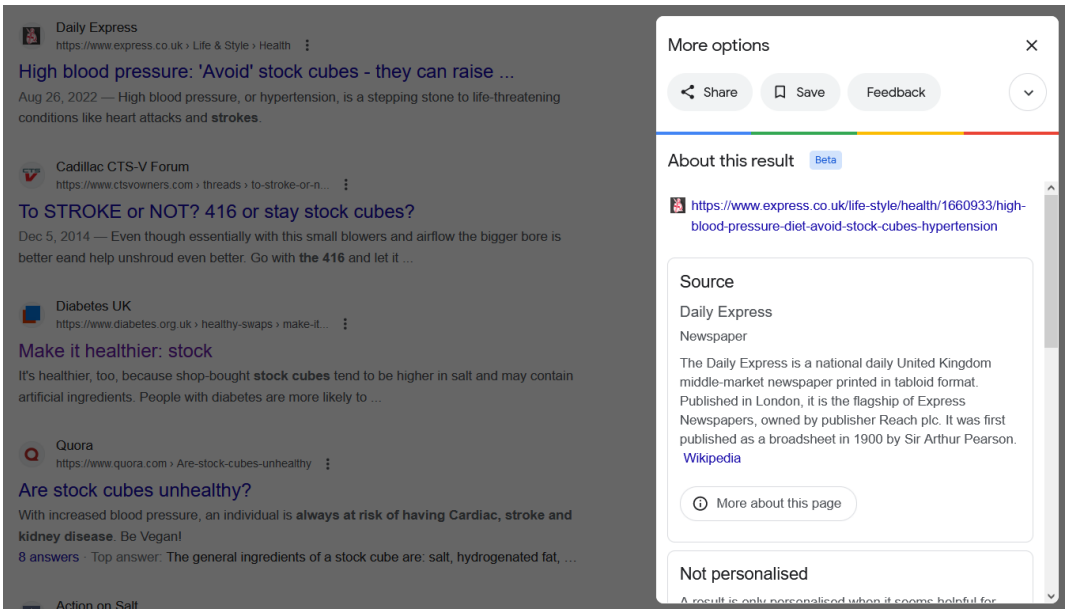

Figure 8: Additional information about your search.

7. You can use all of these tips together to perfect your searches!

1.3 Rewards

Current XP: 20

You are now fueled with knowledge to critically assess information that you come across, whether that is in the form of a social media post, a news channel or a YouTube video. **Remember** the **main media literacy principles**... They will surely **help you in the future**.

Combine this with knowing **how to skillfully Google search** and some might even say that **you have mastered the internet**.

1.4 Summary

Table 1: The main media literacy principles.

Principle	Description
Question Everything	Always carry healthy skepticism. Question the information you come across. Who created it, and what might be their ulterior motive or bias?
Understand the Source	Assess the credibility of the source. Is it a news organization, a blog, or a post? Reliable sources have standards and sources for fact-checking.
Recognize Bias	Every piece of media has a perspective, whether it's political, cultural, or commercial. Understanding the bias will help you interpret the content more accurately.
Verify Information	Don't believe everything you read or see right away. Cross-reference information with multiple sources to verify its accuracy or to identify the bias.
Be Aware of Clickbait	Content designed to grab your attention and generate clicks or views. Be cautious of sensationalist headlines, what reactions or emotions is it trying to provoke?
Media Literacy is Not Just About News	Media literacy extends beyond traditional news media. It includes all forms of media: advertising, entertainment, social media and user-generated content.
Confirmation Bias	Recognize that social media and search engines tailor content to your preferences and beliefs. You are mostly served content that fits your narrative.
Privacy and Security	Protect your information and avoid falling victim to online scams: do not accept all cookies, install an ad-blocker on your browser, do not click every link, do not give away your information to random websites, *etc.*
Cultural Awareness	Recognize cultural differences and how they influence the context. What is acceptable or offensive in one culture can differ from others, including your own.
Ethical Media Consumption	Be respectful online and avoid spreading false or harmful information. If you are feeling frustrated or sad, consider logging off for the day.
Verify the Date	Always check when a publication was created. Outdated information can be misleading or irrelevant.

Google search tips:

- **Tip 1:** Highlight the keywords.
- **Tip 2:** Use quotation marks to get results with an exact match.
- **Tip 3:** To search results in a specific site, write site: behind the keywords.
- **Tip 4:** To remove results with a given expression, place a minus sign behind it, -.
- **Tip 5:** Scroll!
- **Tip 7:** Click the three dots next to a search result to know more about the source.
- **Tip 6:** Combine these tips to achieve the best results.

2 GO DO (I)T!

2.1 Mission description

Godot is a newer, but nevertheless popular, game engine that integrates tools to develop both 2D and 3D games. These games can be made to be played on multiple platforms, such as computers running Windows, macOS or Linux, web browsers, and Android or iOS phones and tablets.

There are other game engines, all of which have unique characteristics that game developers need to take into account when choosing the most appropriate one for their projects. It all depends on what the project requires, for example:

- Does it support 2D and/or 3D games?
- Do developers have access to good, updated sources of information to study and learn from?
- I need a lot of objects in my game, does the engine deal well with that or do people frequently report that it is slow and buggy?
- All the objects in my game need to know if they crashed against one another, does the engine efficiently support collision between shapes?

Among many others, the most common and known game engines are Unity, Unreal Engine, Cry Engine and GameMaker. All of these are great for building anything from simple to complex and big games! However, they also come with a steep learning curve[1].

[1] The overall learning experience, from the initial challenges and struggles to the point where a person becomes skilled and efficient in a given area.

To first dip your toes in the game development pool, Godot is a great option! It is compact, versatile, has great documentation and the programming language it offers for developers to create the logic and behavior of their games is simple to read and write. Finally, Godot **requires no installation**, so let's go do it!

2.2 Goals

This mission calls for you to download Godot to your computer and prepare it to run a game that you will later learn how to develop and make it your own!

2.2.1 Download Godot

1. To download Godot to your computer, click one of the links below, depending on what operating system you are using. The links will lead you to the **download page of the official Godot Engine website**:
   - If you are using **Windows**, click here.
   - If you are using **macOS**, click here.
   - If you are using **Linux**, click here.
   - If you have Steam installed on your computer, you can get the Godot game engine straight from the store.
2. Once you have landed on Godot's website download page, **left-click** on the first, blue button (see Figure 9) and wait for the download to finish.

Figure 9: Download Godot to your computer.

3. Go to the Downloads folder on your computer and search for a zipped (.zip) file for Godot. **Right-click** it and extract all contents (see Figure 10).

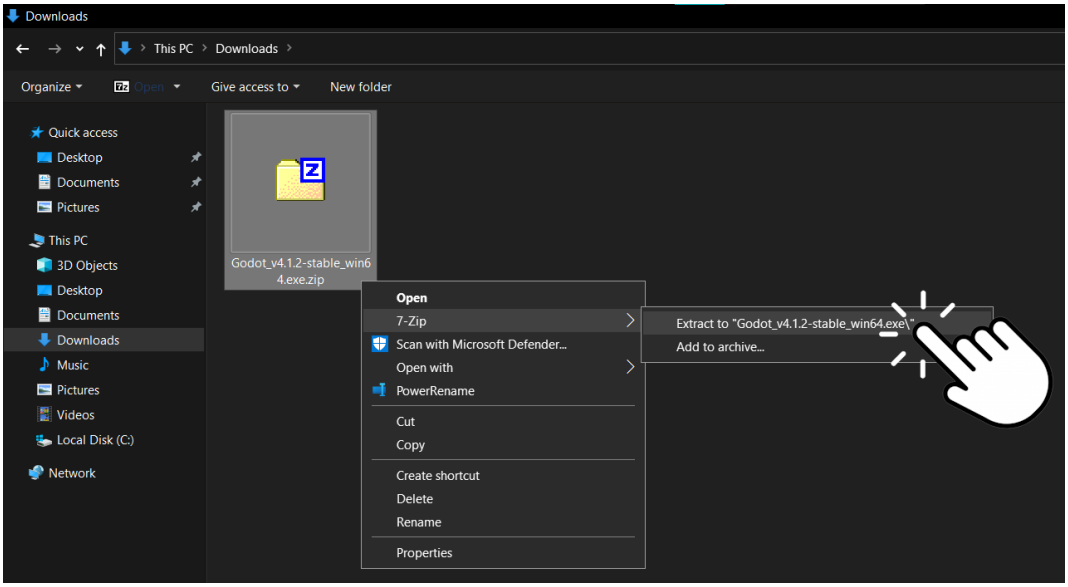

Figure 10: Extract the downloaded file.

4. All done! Open the new folder that was created on your Downloads and **double-left-click** the first executable (the one that **does not** have the word *console* in the name - see Figure 11).

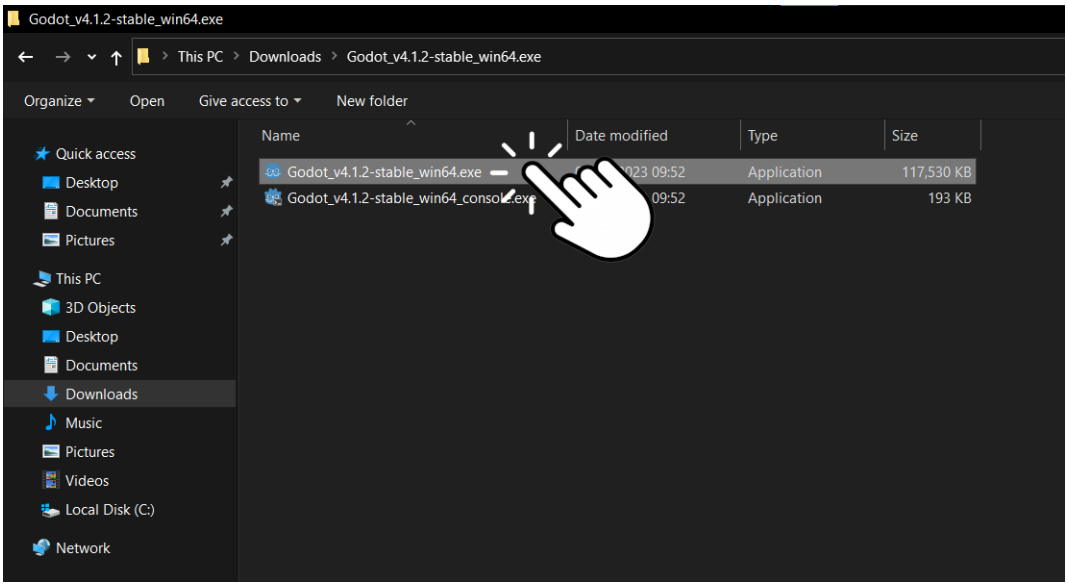

Figure 11: Open the Godot game engine.

2.2.2 Open a Project

Godot is impatient! When you first open it, it promptly asks if you would like to open a project. Luckily for you, **this Toolbox includes a project** that was prepared beforehand, complete with all that is necessary for it to run smoothly.

1. **Left-click** the button that says **Open Asset Library** (see Figure 12).

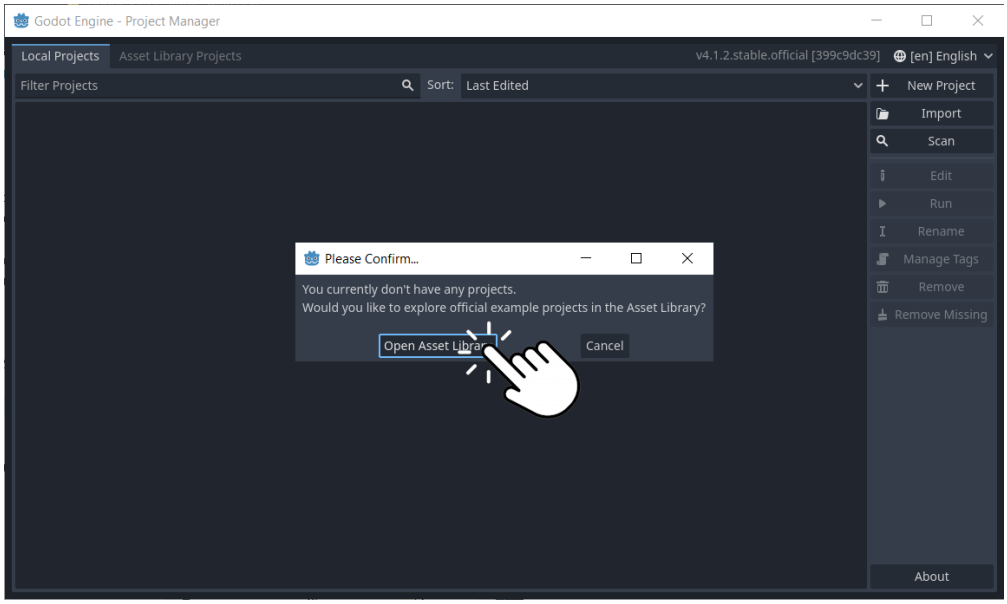

Figure 12: Open Godot's asset library.

2. **Left-click** in **local projects**, top-left corner, and then **left-click** the **Import** button (see Figure 13).

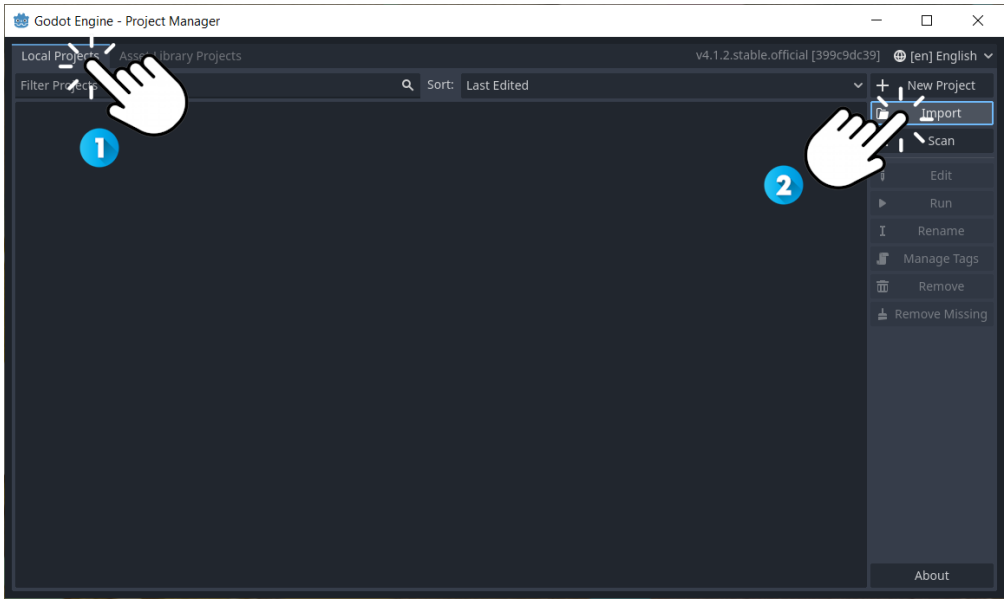

Figure 13: Import a local project.

3. **Left-click** browse projects (see Figure 14).

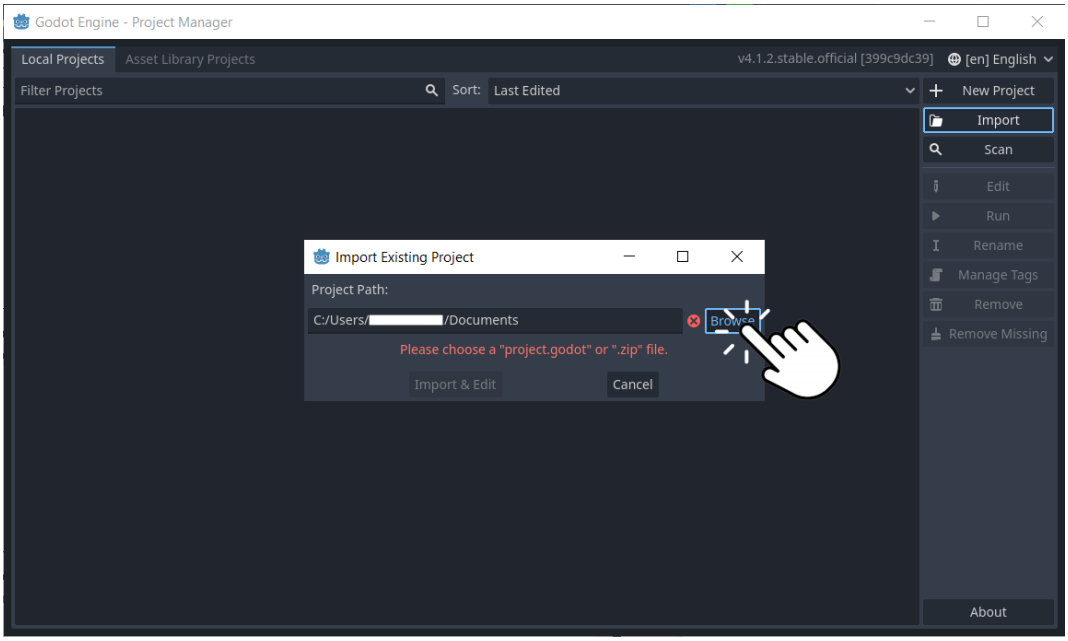

Figure 14: Browse through the files on your computer.

4. **Left-click** on the upwards arrow in the left corner to move upwards in the folder system, this might facilitate searching for the **Game-It-Yourself Toolbox folder**. (see Figure 15).

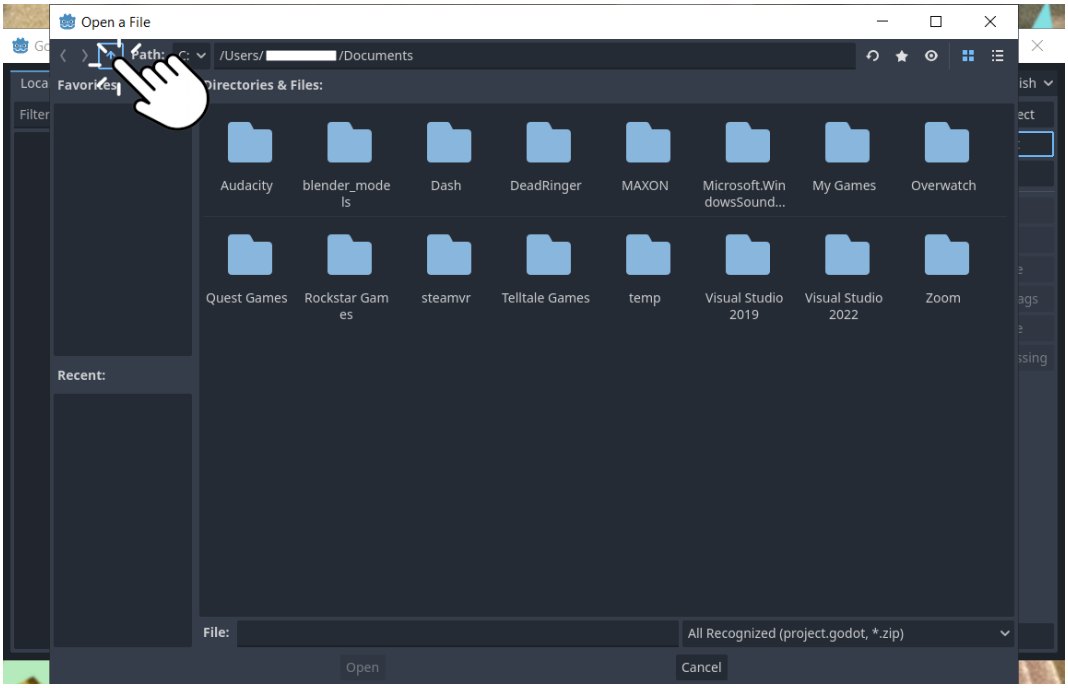

Figure 15: Search for the **Game-It-Yourself Toolbox** folder.

5. Search for the **Game-It-Yourself Toolbox folder** in your computer files and, when you find it, continue to **Level 2 - Poll of Lies → game-it-yourself-poll → Poll of Lies**.

   You will find a **project.godot**, **double-left-click** it (see Figure 16).

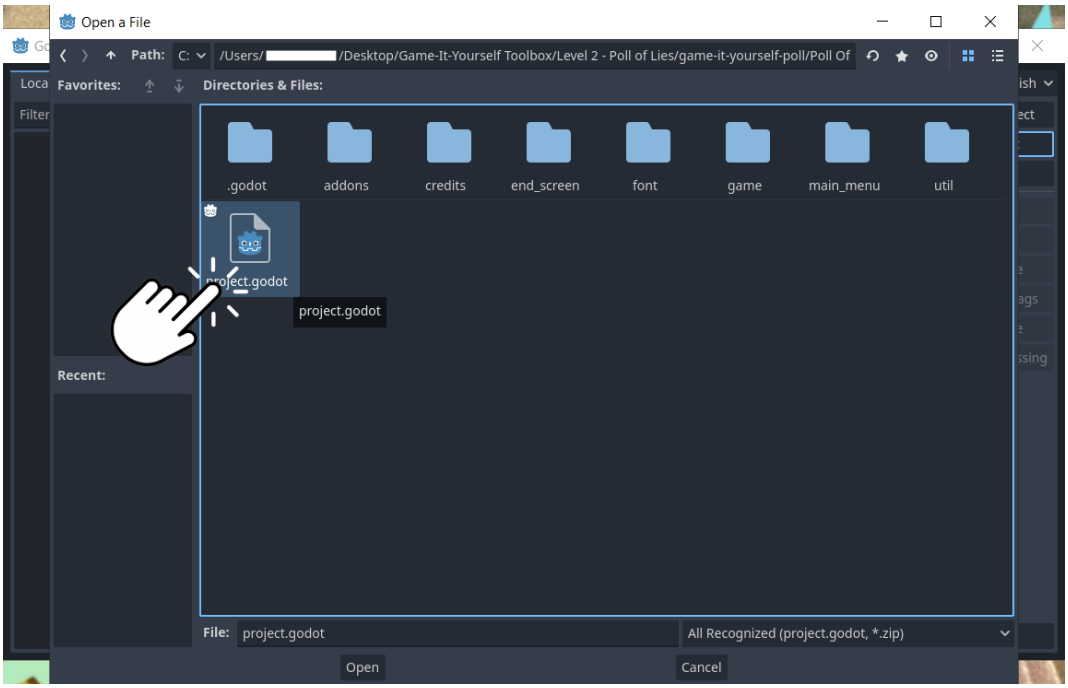

Figure 16: Open the **Poll of Lies** Godot **project**.

6. Godot will ask if you want to **Import and Edit**, **left-click** that button (see Figure 17) and we are ready to rumble! *Shh...* Godot is now opening!

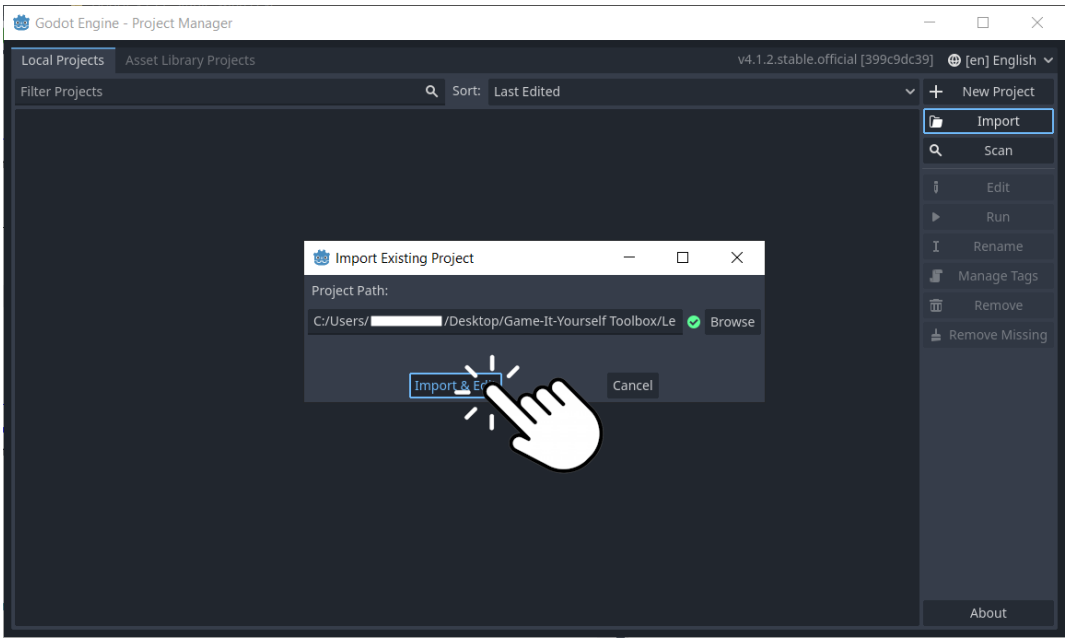

Figure 17: Open the **Poll of Lies** Godot **project**.

2.3 Rewards

Current XP: 35

You are now the proud owner of Godot! By completing this second mission, you became instantly ready for many future projects! For now, let's focus on trying to develop a **Poll of Lies** (queue the dramatic music and evil laughter).

3 Explain Godot to Me Right Now or I Quit

3.1 Mission description

Alright... well, fair enough. What are you looking at right now? Let's unpack **Godot's interface**, it is not that complicated.

Ah! But do not forget that if you ever feel the need to explore further or are left with a lingering question that is itching your brain, you should always check Godot's updated documentation. The Godot Docs website has a search bar on the left side of the page, for easier access to information. Besides this, there is also an official Godot active forum where developers leave their issues/questions and others try to help! Finally, for questions more related to programming, Stack Overflow is the way to go.

Your problem is never unique, someone already went through the same struggle you might be going through and the answer is lying somewhere in the depths of the internet! If you are more of a visual person, YouTube might be the way to go since there are tutorials for everything.

3.2 Goals

3.2.1 Godot's Interface

Godot's editor interface contains everything that is essential for the development of a game. There are 10 main areas of interest, which we will unpack together for you to settle in (see Figure 18):

0. **Viewport:** Where the magic happens! It's here that you see and edit your game.

1. **Main menus:** Quick access to some key scene functionalities, project settings and documentation.
2. **Workspaces:** Choose the view of your scene: 2D, 3D or script editor.
3. **Playtest buttons:** These buttons allow you to test your game.
4. **Open scenes:** The scenes you have open.
5. **Toolbar:** Change between normal cursor, moving objects, scaling, and other options for your editor.
6. **Filesystem:** This is where all the files of your project are: scenes, images, sounds, scripts, *etc.* You can also add new files here.
7. **Scene hierarchy:** It shows all the objects that compose your scene, which you can rearrange in order, add or delete, make (in)visible or add scripts to.
8. **Inspector:** Every object in your scene has properties, like size and color, all of which can be inspected and changed here.
9. **Bottom panel:** This allows you to check how your game is behaving, through the output and debugger tabs, or do some cool animations.

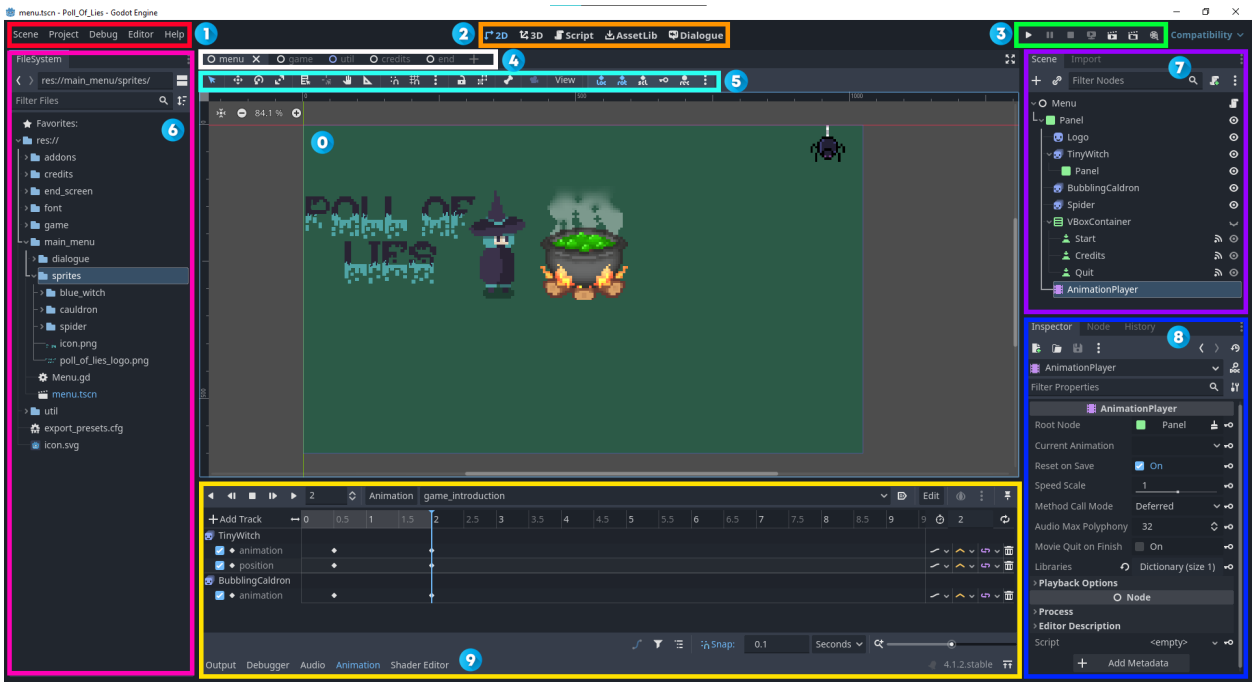

Figure 18: Godot's interface.

3.2.2 Godot's Scenes and Node-based System

A game in Godot is made up of different scenes, each with an environment to explore, characters to talk to, events to unfold and things to collect - much like a scene in a play! Every one of these elements is a **node** in Godot. A node can be anything from an image that represents an object or a character, to a 3D body with movement that collides with walls, and walls will also be nodes!

A **scene** is a **base node composed of a hierarchical structure of more nodes**. Confusing at first, yes, but you will explore it in just a bit. It is easy! Remember that the documentation serves to inform you of everything you are feeling unsure about.

To control the behavior of a scene or node you can create a script for them and, in it, you program to instruct how everything works. To program, Godot uses **GDScript**, a scripting language specifically created for the engine. It is straightforward and comfortable to learn and understand, almost like writing and reading in English. One thing that you should be aware of: GDScript is **indentation-sensitive**, so where and how you place your code in Godot is very important. Indentation, like at the beginning of a paragraph, is a displacement of the line or block of code to the right via tabs (a key in your keyboard, normally close to the left shit, control and capslock). If the code shown to you is indented, try to write it or copy it exactly as is so it all works perfectly.

3.2.3 Run It... If You Can!

This is quite a simple task, **start the game**! Either press the first of the playtest buttons (Figure 18), or **press F5**, and enjoy your playthrough.

Wait... **WHAT?!** The game is not running?

3.2.4 Call me Handy Manny, I'll Fix It Myself!

Well, it seems the developers forgot to put the FINAL_actuallyFINAL_last_version of the game inside the Toolbox folders! The first thing that might have caught your attention when trying to play this new, favorite game of yours, is that **no button works**. That will be an easy fix!

1. Open the Menu scene and turn on the visibility of the buttons by clicking on the eye next to the VBoxContainer node. Now you can see them, **they are there**, so why don't they work? (See Figure 19).

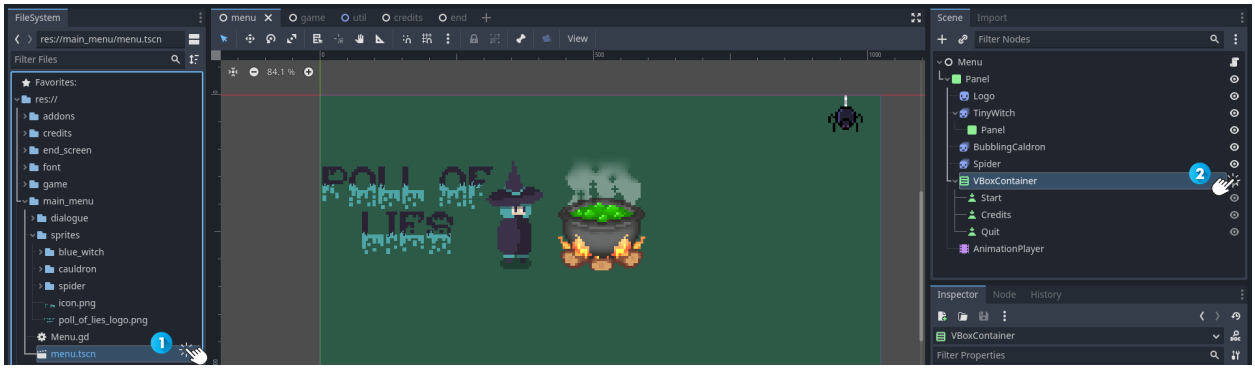

Figure 19: Turn on the buttons' visibility.

2. Click on the Start button and check its node properties.

   Godot has an integrated **signal system** that allows for messages to be sent from one object to the other when something special happens, allowing for the appropriate answer to occur. In this case, we need **the button to signal to the main menu that it** (the button) **has been pressed** in order to ACTUALLY start the game. Click the `pressed()` signal and then Connect... (see Figure 20).

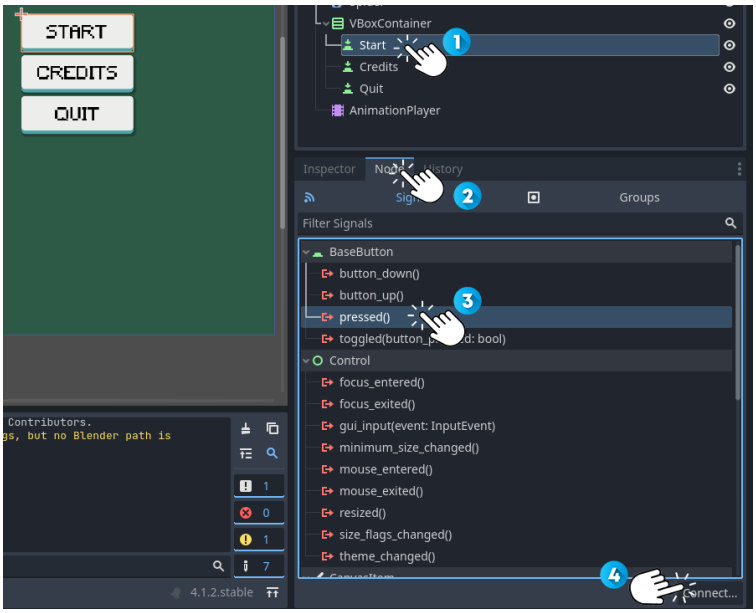

Figure 20: Connecting the Start button's pressed signal.

3. Now you can **choose where this signal connects to**, as well as what to call the Receiver Method (this name will be important for the next steps). Through this approach, you can only connect the signal to one of the other nodes inside the same scene where the button is. In this case, since we need the button to *tell* or message the Menu that it has been pressed, simply connect it as is, to the Menu (see Figure 21).

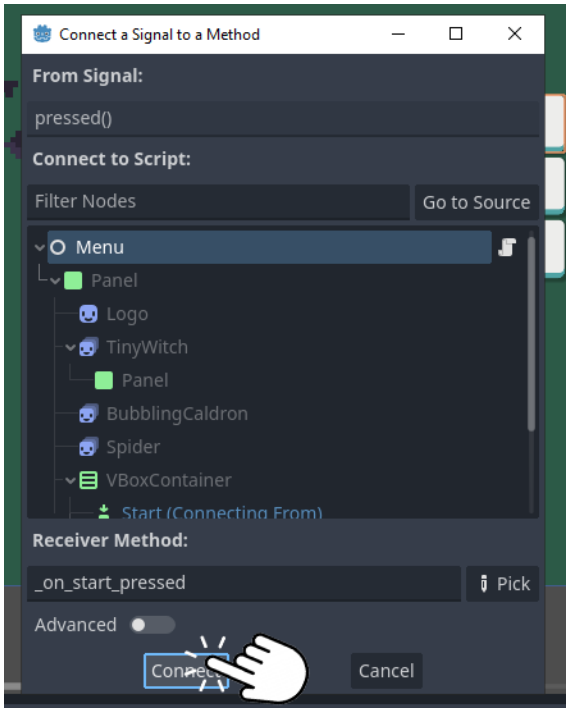

Figure 21: Connect the signal `pressed()` to the Menu node, via the `_on_start_pressed` method.

4. You have been teleported to the world of programming! The line of code in front of you has the same name as the Receiver Method that you selected when connecting the signal.

   A **method**, or **function**, is a reusable block of code where programmers instruct the program, in this case a game, on how to perform a certain task. You are in charge of **programming how the Menu scene behaves when the Start button is pressed**. Note that there is a **green symbol** behind this method's name. To better understand what is happening, press it! Never be afraid to explore the tools, CLICK EVERYTHING (see Figures 22 and 23).

Figure 22: Check the information about a signal connected to a method.

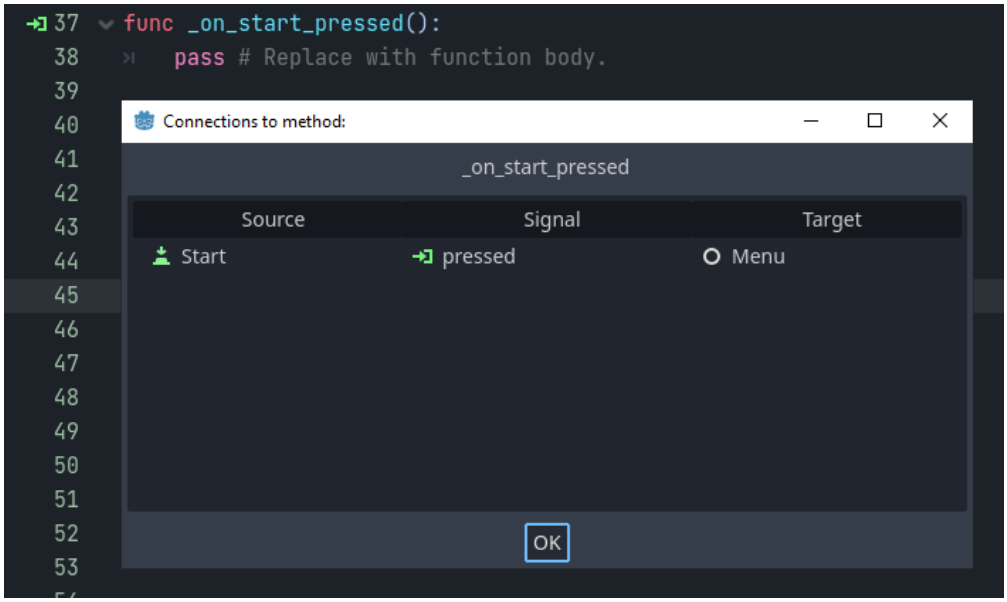

Figure 23: Understand the source and target of a signal connected to a method.

   See how, by clicking the green symbol, you are informed about where the signal is sent from, what the signal is and where it goes.

5. The actual gameplay takes place in a scene called Game. To facilitate the organization of the program, a different scene, Util, is in charge of knowing where everything is, which is a good practice in programming[2]. Look at the script Util.gd: the first path that appears

[2]**Single-responsibility principle:** each node should only have one "job", so the Menu should not have to worry about where the Game scene is located. However, Util's function is exactly that: to know about every other scene and what might be useful to them. Furthermore, if the location of the Game scene changes, you only have to alter it in one script! Work smarter, not harder. [1]

(line 4) stores the information about the location of the Game scene (see Figure 24). Util is a scene that is automatically initialized when the game starts. If you want to know more about how the developers did this, check out Godot's documentation on Singletons (Autoload).

Go back to the Menu.gd script. To instruct the Menu to change to the Game scene when the button is pressed just **write some code**!

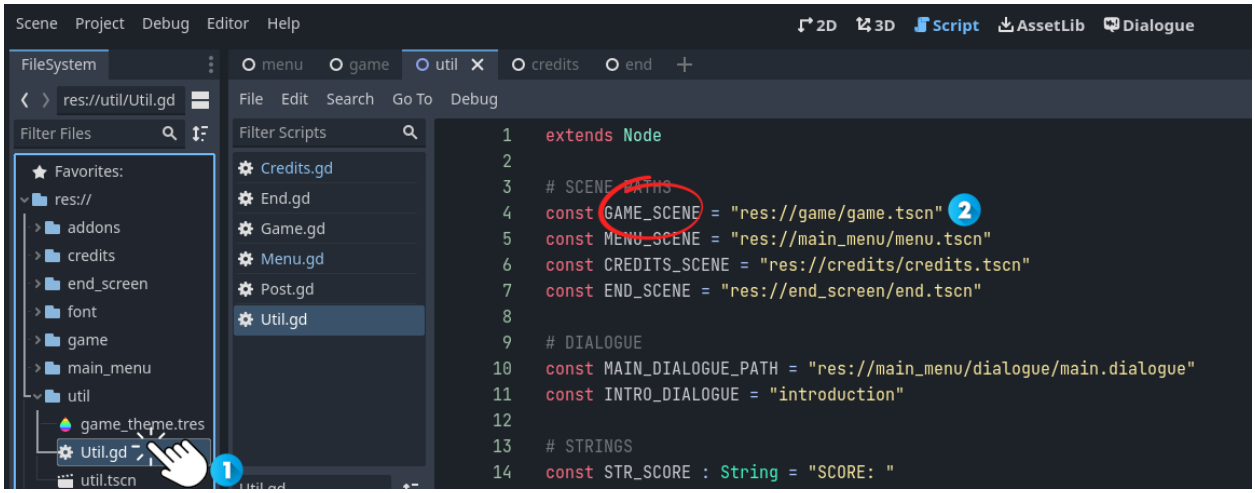

Figure 24: Script Util.gd stores the path to the Game scene.

Godot has a native method to change between different scenes: `get_tree().change_scene_to_file()`. This method receives an argument inside the parenthesis to know where to change the scene to. Arguments, if necessary, are passed to a function to give it some information to process.

Listing 1: The `on_start_pressed()` function in the Menu node.

```
func _on_start_pressed():
    get_tree().change_scene_to_file(Util.GAME_SCENE)
```

We are passing the argument `Util.GAME_SCENE` to this method by placing it inside the parenthesis. The script Util, Figure 24, has a variable named `GAME_SCENE` with the path to this scene, but to call the variable in another script we have to specify where it comes from, hence `Util.GAME_SCENE`.

After making these changes, try to run the game again and check if the Start button works already. IT DOES?! Brace yourself and attempt to make the other buttons work using the information that has already been provided. To understand how to close the game when the Quit button is pressed, check out Godot's documentation on Handling quit requests.

Remember to **deactivate the visibility** of the VBoxContainer that contains the Menu buttons when you are done. This way, the buttons will be visible only when the player interacts with the witch, it builds up suspense! (See Figure 19).

3.2.5 The Game is EMPTY

WHAT?! Not again! It's starting to sound like the developers left the game incomplete only for you to put it back together and learn how to develop games! WHY WOULD THEY DO THAT?

As the witch told you, the goal of the game is simple: between two social media posts shown, choose the one that you believe is true. If you are unsure as to which one that is, search the internet for the answer using the tips you have already learned. However, the game is not showing anything! To fix this, we have to understand what is going on with the code that already exists...

1. Through the FileSystem tab in Godot's interface, go to the Game folder and open the game.gd script (see Figure 25).

```
1  extends Node
2
3  @onready var _animation_player = $AnimationPlayer
4  @onready var _left_post = $MarginContainer/VBoxContainer/HBoxContainerPosts/LeftMarginContainer/LeftPost
5  @onready var _right_post = $MarginContainer/VBoxContainer/HBoxContainerPosts/RightMarginContainer/RightPost
6  @onready var _score_label = $MarginContainer/VBoxContainer/HBoxContainerScore/MarginContainer/ScoreLabel
7  @onready var _fail_attempt_array = [
8      $MarginContainer/VBoxContainer/HBoxContainerScore/MarginContainerFail3/Fail3,
9      $MarginContainer/VBoxContainer/HBoxContainerScore/MarginContainerFail2/Fail2,
10     $MarginContainer/VBoxContainer/HBoxContainerScore/MarginContainerFail1/Fail1
11 ]
12
13 var score : int = 0
14 var number_failed_attempts : int = 0
15
16 # Called when this scene starts
17 func _ready():
18     # Stop music from menu and start game music
19     SoundManager.instance.stop_all_audio()
20     SoundManager.instance.play_game_music()
21
22     reset_variables()
23
24     # Connect to signals of when player selects the correct post
25     assert(_left_post.scored_point.connect(update_score) == OK)
26     assert(_right_post.scored_point.connect(update_score) == OK)
27
28     # Connect to signals of when player selects the incorrect post
29     assert(_left_post.failed_attempt.connect(update_fail_attempts) == OK)
30     assert(_right_post.failed_attempt.connect(update_fail_attempts) == OK)
31
32     # Show score to the player
33     _score_label.text = Util.STR_SCORE + str(score)
34
35     # Put images on the right and left post
36     assign_random_images_to_posts()
```

Figure 25: Beginning of the game.gd script.

2. Godot has a special method for all nodes called `_ready()` where programmers control what happens at the beginning of each scene.

   The grey lines are comments and these are ignored by the game when it runs. It is a good practice to write some explanations through comments so other programmers can understand what the thought process was when the code was written (even better when

you use it to remember your own logic after weeks of not opening Godot). At the very end of the `_ready()` function, it seems that the programmers call another method that is supposed to put images...

Look for the `assign_random_images_to_posts()` method in this same script!

3. OooOoOOHh! By reading the comments, it looks like this method is supposed to assign a random image to a post, and the chance of this image being true or false is 50-50% (see Figure 26).

```
39 func assign_random_images_to_posts() -> void:
40     # 50-50% random chance for the posts to either be true or false, so it doesn't get predictable
41     # Remember that when one is true, the other has to be false!
42     pass
43
44
```

Figure 26: `assign_random_images_to_posts()` method in the game.gd script.

There are many ways of doing this, one of which is to create a random floating-point number between 0 and 1, as there is an equal chance of the number either being greater or smaller than 0.5! Godot has a method for creating a number just as this: `randf()`. We can do one thing if the number is greater, and do another if it is smaller, so... random. Delete the line that says `pass` and get this working!

Listing 2: Generate a random number between 0 and 1 and check if it is greater or smaller than 0.5 inside the `assign_random_images_to_posts()` method.

```
func assign_random_images_to_posts() -> void:
    # 50-50% random chance for the posts to either
    # be true or false, so it doesn't get predictable
    # Remember that when one is true, the other has to be false!

    # Create a random number between 0 and 1
    var chance_true_or_fake = randf()

    # If the number is greater than 0.5,
    # the left post will be false and the right will be true
    if (chance_true_or_fake > 0.5):
        # something
    else:
        # something else
```

First, the random number is generated by calling the `randf()` method and the result is stored in a variable called `chance_true_or_fake`. After this, the code checks if this number is bigger or smaller than 0.5 through an if-else condition. An if-else condition block evaluates an expression, in this case, `chance_true_or_fake > 0.5`, which can either be

true or false, and it only runs the code that is inside the true part. For example, if the number is **not** bigger than 0.5, the expression inside the if condition parenthesis is false, therefore only the code inside the else block executes.

The programming gods have decided that if the number is greater than 0.5, the post shall be false, and true it shall be if the number is smaller!

4. But wait! Just below this method are two others that seem interesting (see Figure 27)... They return random images, but from where and to where? We have to follow the leads!

```
45  # These methods work as follows:
46  # Since we don't know exactly how many items are inside the array,
47  # we can create a random number between 0 and the number of items in the array
48  # which will correspond to a random item of the array.
49  # This is done like so: randi() % Util.FAKE_POST_ARRAY.size()
50  # randi returns a VERY BIG number from 0 to 4294967295 (inclusive),
51  # but by dividing it (%) by the number of items in the array, we get a random index!
52  # the pop_at() method for an array simultaneously returns and removes an item from the array
53  # so after an image is chosen it can't be chosen again!
54  # Passing the random index to the pop_at() method for the array does the trick.
55  func get_random_fake_post():
56      return Util.FAKE_POST_ARRAY.pop_at(randi() % Util.FAKE_POST_ARRAY.size())
57
58  func get_random_true_post():
59      return Util.TRUE_POST_ARRAY.pop_at(randi() % Util.TRUE_POST_ARRAY.size())
```

Figure 27: `get_random_fake_post()` and `get_random_true_post()`

5. To understand how the images are assigned to a post, click one of the empty, darker blue spaces you see in your Game scene and then click on the Post's script through the scene hierarchy (see Figure 28).

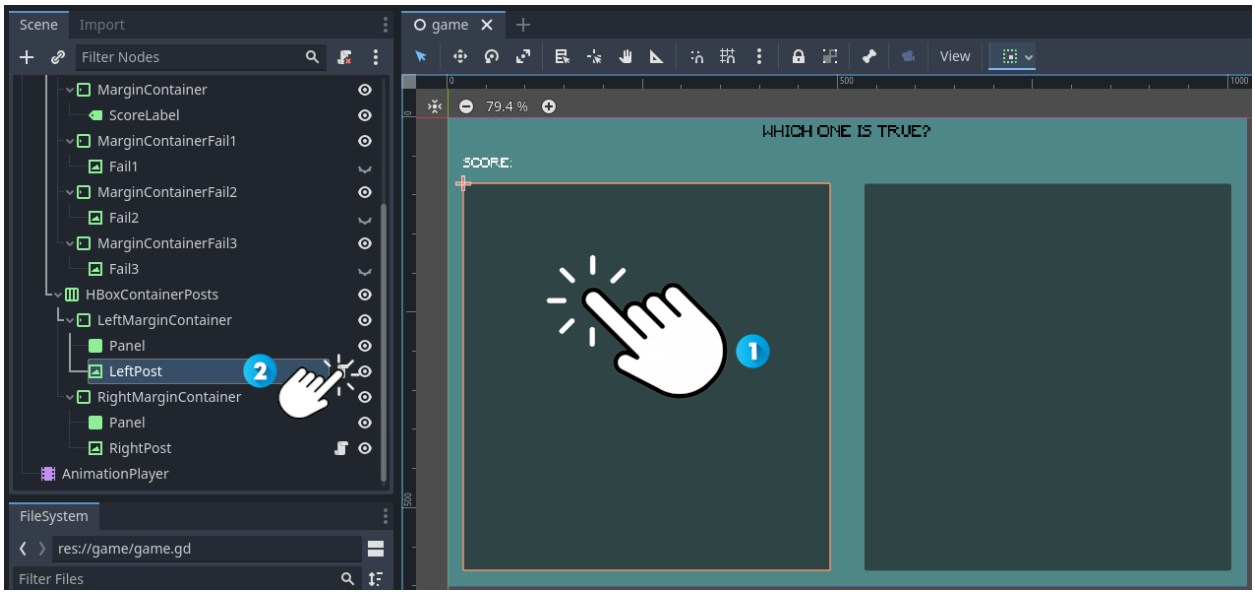

Figure 28: Go to the `Post.gd` script.

The `assign_image(new_texture: Texture2D, new_bool: bool)` method in Post.gd seems promising (see Figure 29).

This function receives one argument that is a Texture2D, so an image, and another that is a bool, short for boolean. A boolean is a type of variable that can only either be true or false.

It seems that in order to assign an image to a Post we have to use this method and pass it both an image and a boolean. The image will be what is shown to the player and the boolean tells the Post if the image is true or not.

```
1   extends TextureRect
2
3   signal scored_point
4   signal failed_attempt
5
6   var _is_post_true : bool
7   var _mouse_hovering : bool
8
9   func _ready() -> void:
10      # Connect self to mouse entered and exited signals
11      assert(mouse_entered.connect(_on_mouse_entered) == OK)
12      assert(mouse_exited.connect(_on_mouse_exited) == OK)
13
14  # Method that receives a Texture2D and a boolean
15  # The Texture2D is the image that is going to be placed in this post
16  # The boolean tells the post if it is true or false
17  func assign_image(new_texture : Texture2D, new_bool : bool) -> void:
18      texture = new_texture
19      _is_post_true = new_bool
20
```

Figure 29: Post's script method `assign_image()` takes care of showing the images.

6. You are probably scratching your head nervously and thinking: *but where do these images come from?* Luckily, it appears to be that the developers left that part taken care of. Go to Util's script, `Util.gd`, we are almost there!

```
48  func _init() -> void:
49      FAKE_POST_ARRAY = []
50      TRUE_POST_ARRAY = []
51
52      FAKE_POST_ARRAY_COPY = []
53      TRUE_POST_ARRAY_COPY = []
54
55      # Get all the images from the fake post folder onto the fake post array
56      _load_array_with_image_paths(FAKE_POST_ARRAY, FAKE_POST_PATH)
57
58      # Get all the images from the true post folder onto the true post array
59      _load_array_with_image_paths(TRUE_POST_ARRAY, TRUE_POST_PATH)
60
61      TRUE_POST_ARRAY_COPY = [] + TRUE_POST_ARRAY
62      FAKE_POST_ARRAY_COPY = [] + FAKE_POST_ARRAY
```

Figure 30: Util's image arrays.

Util has two arrays with the images ready to use, one for the true images and one for the fake ones.[3] Arrays are a type of data structure in programming that holds data, such as numbers or booleans, in an organized manner, much like a folder. Each item in an array has a position or index. The first item has an index of zero (0), and subsequent items have increased indexes by one (1).

7. Oof, go back to the game's script, `game.gd`. Now we can use the information we have gathered to complete that `assign_random_images_to_posts()` method! A quick recapitulation:

   (a) If the random number we generated between 0 and 1 is greater than 0.5, the left post is false and the right post is true. As such, if the number is not greater than 0.5, the left post is true and the right post is false;

   (b) To assign an image to a post we use the `assign_image(new_texture: Texture2D, new_bool: bool)` method;

   (c) When calling this method we have to pass it an image and a boolean;

   (d) Util has the true images stored in the `Util.TRUE_POST_ARRAY` and the fake ones in `Util.FAKE_POST_ARRAY`;

   (e) `game.gd` has methods to retrieve images from both of these arrays: `get_random_fake_post()` and `get_random_true_post()`;

   (f) To assign a true image with the `assign_image()` method, the first argument passed should be an image from the `Util.TRUE_POST_ARRAY` and the second argument should be `true` (boolean), and vice versa.

[3] You can explore how these arrays came to be by looking at what `_load_array_with_image_paths()` method does. It goes through two of this project's folders, game/posts/fake and game/posts/true, and loads all images to the correct array when the game is initialized.

Listing 3: Completion of the `assign_image()` method in the `game.gd` script.

```
func assign_random_images_to_posts() -> void:
    # 50-50% random chance for the posts to either
    # be true or false, so it doesn't get predictable
    # Remember that when one is true, the other has to be false!

    # Create a random number between 0 and 1
    var chance_true_or_fake = randf()

    # If the number is greater than 0.5,
    # the left post will be false and the right will be true
    if (chance_true_or_fake > 0.5):
        _left_post.assign_image(get_random_fake_post(), false)
        _right_post.assign_image(get_random_true_post(), true)
    else:
        _left_post.assign_image(get_random_true_post(), true)
        _right_post.assign_image(get_random_fake_post(), false)
```

That one made you sweat a little bit, uh? **Do not get discouraged by programming if you cannot understand every single thing that was done to achieve this result.** Programming is this going back and forth to see how different pieces fit together, or not. With practice and without fear of failure, anyone can program. These exercises are also good for the imagination... If such a simple game is this "complicated", have you imagined how games like Marvel's Spider-Man 2 (2023)[4], The Legend of Zelda: Tears of the Kingdom (2023)[5] or Minecraft (2011)[6] were created? It is so impressive! HOW EXCITING, let us keep going!

3.2.6 Stop Stealing My Score!

Eheh, sorry. We were keeping your score so we could claim it as our own in case you reached some sort of new all-time mega record. Ok... You can fix it so it will not happen again.

1. Hm... There seems to be a method in the `game.gd` script where the score is updated (see Figure 31). Is it not working? Were is this method called?

[4]Marvel's Spider-Man 2. Sony Interactive Entertainment (2023) can be found at https://www.playstation.com/pt-pt/games/marvels-spider-man-2/

[5]**The Legend of Zelda: Tears of the Kingdom**. Nintendo (2023) can be found at https://www.nintendo.pt/Jogos/Jogos-para-a-Nintendo-Switch/The-Legend-of-Zelda-Tears-of-the-Kingdom-1576884.html

[6]Minecraft: Bedrock Edition. Mojang Studios (2011) can be found at https://www.minecraft.net/en-us/article/java-or-bedrock-edition

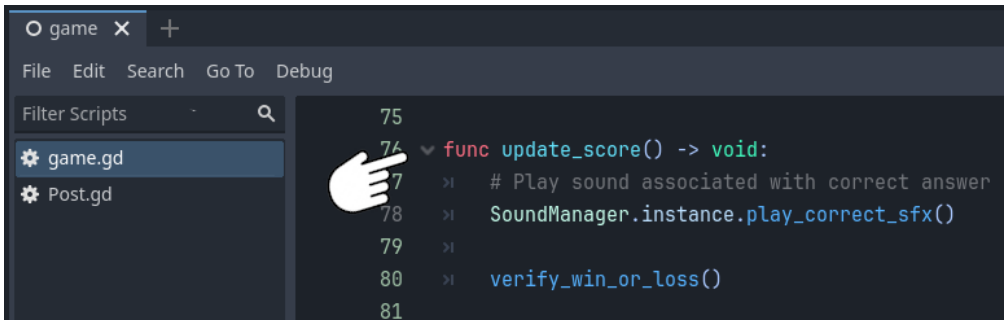

Figure 31: `update_score()` method in `game.gd`

2. A thorough reading of the `_ready()` method of this script tells us that the Game scene connects to a signal from each post, left and right (see Figure 32). What does this mean?

   Well, you have learned how to connect signals through Godot's interface, but it is also possible to **connect signals between different nodes through code**. This tells us that both the left and right posts have a signal called `scored_point`, to which the Game scene is connecting. When this signal is emitted, the `update_score()` method of the Game scene is called.

```
16  # Called when this scene starts
17  func _ready():
18      # Stop music from menu and start game music
19      SoundManager.instance.stop_all_audio()
20      SoundManager.instance.play_game_music()
21
22      reset_variables()
23
24      # Connect to signals of when player selects the correct post
25      assert(_left_post.scored_point.connect(update_score) == OK)
26      assert(_right_post.scored_point.connect(update_score) == OK)
27
28      # Connect to signals of when player selects the incorrect post
29      assert(_left_post.failed_attempt.connect(update_fail_attempts) == OK)
30      assert(_right_post.failed_attempt.connect(update_fail_attempts) == OK)
31
32      # Show score to the player
33      _score_label.text = Util.STR_SCORE + str(score)
34
35      # Put images on the right and left post
36      assign_random_images_to_posts()
```

Figure 32: Game scene connecting to each posts' `scored_point` signals and calling `update_score()` when that signal is emitted.

3. When is the `scored_point` signal emitted? If the signal is sent from the posts, to understand you have to go to the `Post.gd` script. Spoiler: the `_input()` method is where all the fun happens (see Figure 33, line 31)! This method is yet another of Godot's native methods. It is called when an input event happens inside a node, which can be something like

a mouse click, pressing or releasing a certain key, *etc.* **Godot's native methods can be explored further by clicking the blue arrow next to their names** (see Figure 33, line 22).

```
22 func _input(event) -> void:
23     # If player clicks on the correct answer
24     if (event is InputEventMouseButton && event.is_action_released("mouse_click") && _mouse_hovering):
25         # Play click audio
26         SoundManager.instance.play_click_sfx()
27
28         # If the post is correct, emit a signal to increase the score
29         if texture != null:
30             if _is_post_true:
31                 scored_point.emit()
32             # Otherwise, emit a signal to show a new failed attempt
33             else:
34                 failed_attempt.emit()
```

Figure 33: `Post.gd` emits the `scored_point` signal when the player chooses the correct answer.

This `_input()` method is called every time an event happens. When it is called, it checks if that event was the click of a mouse, which means that each time one of the posts is clicked by the player, the code inside the first if-condition runs (see Figure 33, line 24). Inside this if-condition, the post checks if they themselves are true or false, that argument you passed when assigning an image to each post! (see Figure 33, line 30-34) If the post is true, then the signal emits, since the player just scored a point!

4. Going *all* the way back to the Game scene, there is a label to show the score (see Figure 34). A label is a node whose sole purpose is to display text. Great, that is just what is needed!

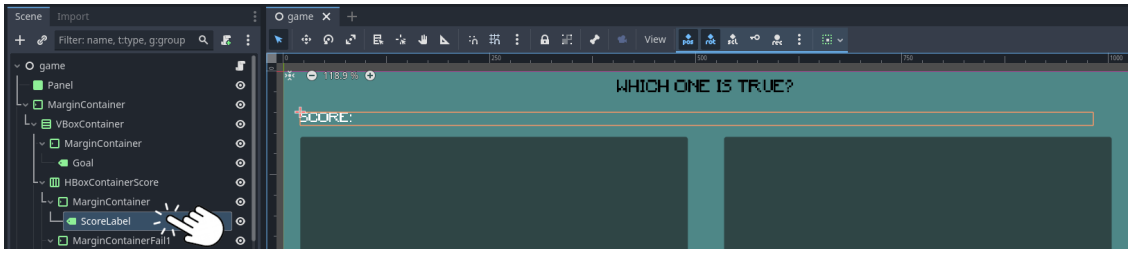

Figure 34: Score label in the Game scene.

In the `game.gd` script, this label is prepared for you to use, as well as a variable to keep track of the player's `score` (see Figure 35).

The variable `score` is an int, short for integer, which is a data type of whole numbers (no decimal places). When created, it is initialized as being zero (0).

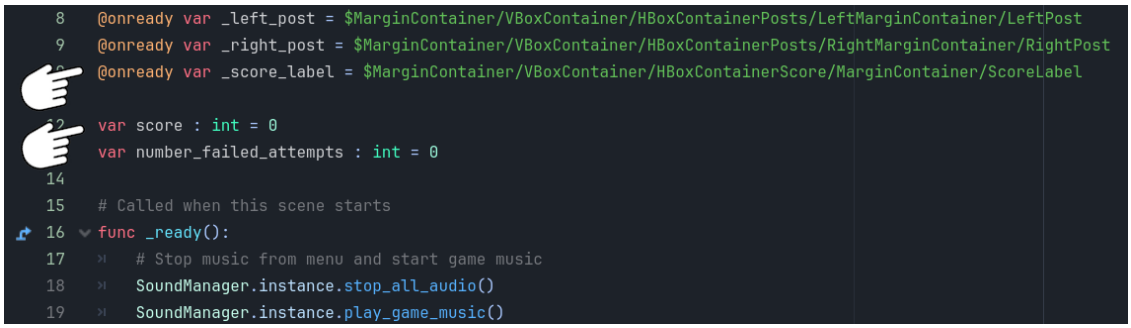

Figure 35: `_score_label` and `score` variable in `game.gd`

So each time the `scored_point` signal is emitted, the `update_score()` method runs and the `score` needs to be increased by one (1).

Increasing a number by one, or any other value, is the same as stating that said number is equal to themselves plus that value, like so: `score = score + 1`. GDScripts facilitates this operation and this sum is the same as: `score += 1`. It is starting to come together!

Listing 4: Increase the `score` by one (1) each time the player gets a correct answer.

```
func update_score() -> void:
    # Play sound associated with correct answer
    SoundManager.instance.play_correct_sfx()

    # If the player answered correctly, increase the score by one point
    score += 1

    verify_win_or_loss()
```

Change your code and try running the game again!

5. Uh?! **It is still not changing?** OH RIGHT! You did not update the text of the `_score_label` after updating the score!

   Quickly now, there is a game waiting to be played!

Listing 5: Update the `score` and the `_score_label` in `game.gd`.

```
func update_score() -> void:
    # Play sound associated with correct answer
    SoundManager.instance.play_correct_sfx()

    # If the player answered correctly, increase the score by one point
    score += 1

    # Update the text to show this new, increased score
    _score_label.text = "SCORE: " + str(score)

    verify_win_or_loss()
```

Each time this method is called, the label's text is changed to show the new score. The way we join two phrases in GDScript is by actually summing them together. However, since `score` is a number, we have to first transform its value into text.

A variable that holds text in programming is called a string, str for short. Thus, to transform the `score` from an integer to a string, do `str(score)`. Say the `score` was equal to `20`. After executing `str(score)`, the `score` becomes equal to `"20"`, get it? So then, the `_score_label` is equal to `"SCORE: " + "20"`, which is `"SCORE: 20"`.

Try your game now and see if it is finally working!

3.2.7 I Like to Spread Misinformation Online

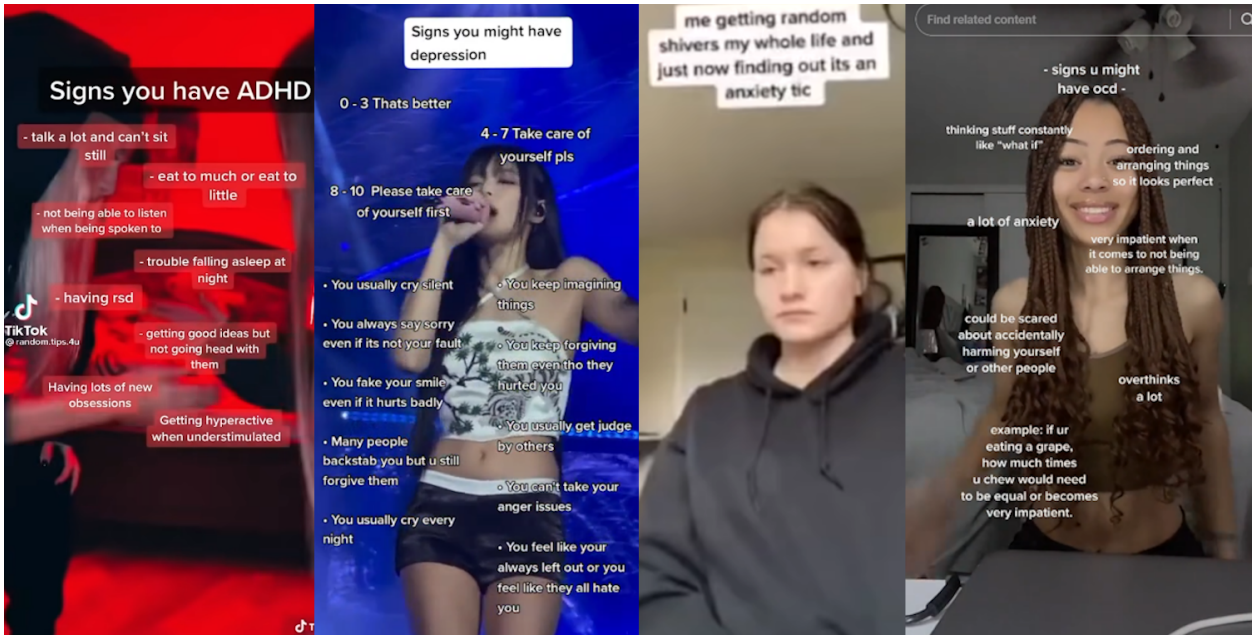

Figure 36: Are these signs that you have a mental illness?

If you are an avid user of social media, you have probably seen videos such as the ones above (see Figure 36). Do you identify any of these traits in yourself?

Misinformation is not always obvious. It sometimes comes off in the form of advice or a helpful tip, even though the information being shared is **incomplete**, **missing context** or simply **false**. The posts above may lead to grave misconceptions about the causes, symptoms, and treatment options for mental health conditions, potentially hindering individuals from seeking appropriate assistance and support. People may share a combination of some of these traits, but having one or two symptoms can not alone predict a mental illness, though it may be cause for seeking help [2], [3].

If you think you could be suffering from a mental illness, talk to the trusted adults around you, like a guardian, a parent of a close friend, a teacher at school or university, or a coach, and try to get professional help from a healthcare provider [7].

Social media platforms enable users to create and share content in a virtual community whilst connecting with others. Inherently, there is nothing wrong with that. However, some people take advantage of the immensity and apparent anonymity of the internet in mischievous ways. Others simply think that they are doing their community a service by speaking what they believe to be true, even though it may be factually inaccurate or incorrect. Today you will be one of those people, **for education purposes**, of course. This is to **understand how easy it is to pretend on the internet**, especially with the rise of artificial intelligence. Zeoob is an educational tool with which you can create almost any type of social media post, with whichever information you want, and make it look convincingly like an authentic post captured in a screenshot.

[7] Help for Mental Illness: https://www.nimh.nih.gov/health/find-help

Create a fake post with Zeoob and save it on your computer! You can add it to the game so other players can later try to guess whether it is fake or not.

Once you have finished creating your post, add it to your game. Since it was fabricated, it can not be considered a truthful post, so add it to the folder of fake images and the game does the rest (see Figure 37). Play the game to see if you can spot it!

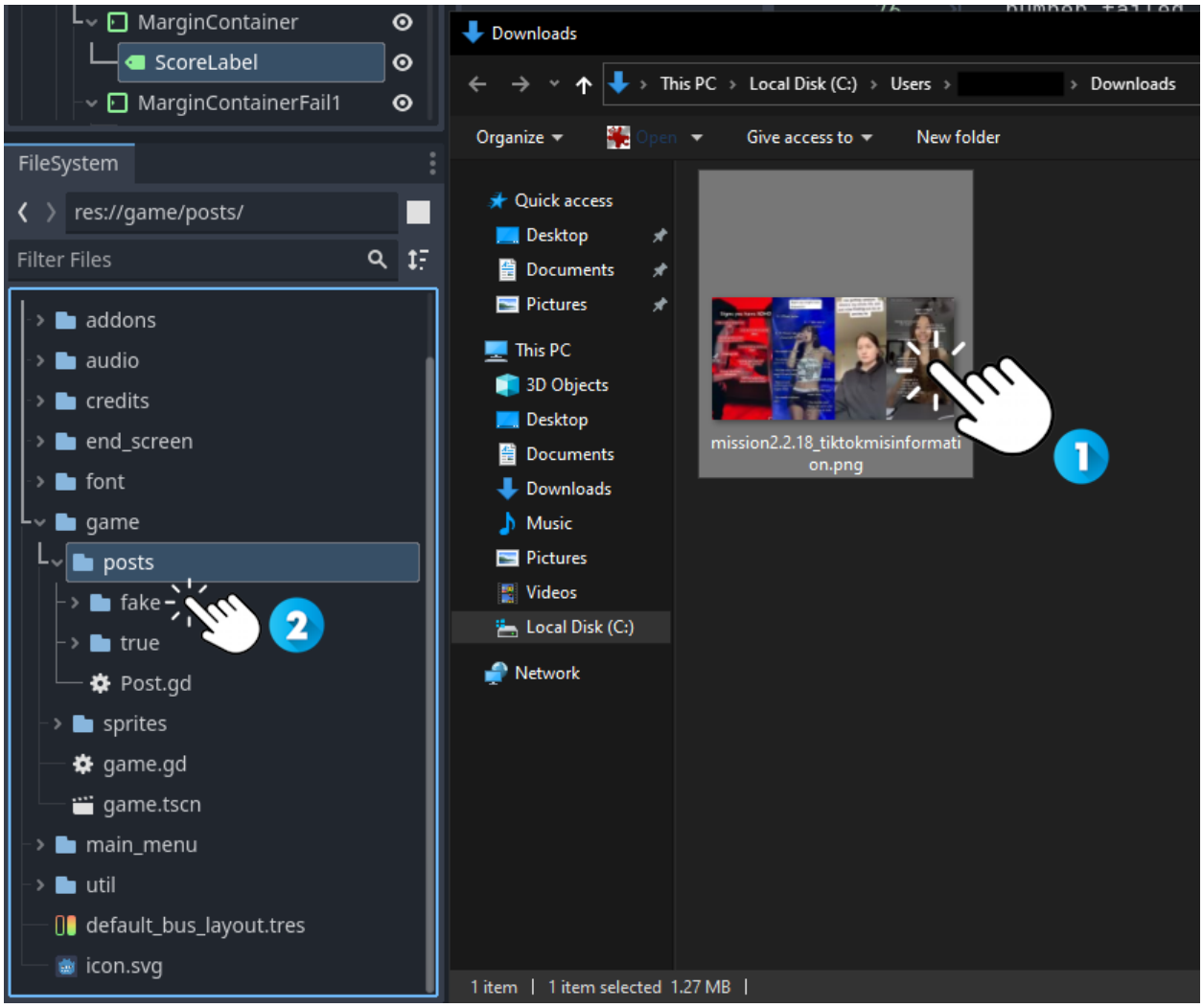

Figure 37: Drag the post you created onto the fake posts' folder.

3.2.8 This Game is Now Mine

What?! Are you trying to steal our game???

Well... It does make sense since you have fixed most of what our developers left unfinished... Alright, you can add your name to the credits. Welcome to the team, [~~REDACTED~~]!

1. Open the Credits scene in Godot and then click the `Developers_Label` to add your name to the team of developers (see Figure 38).

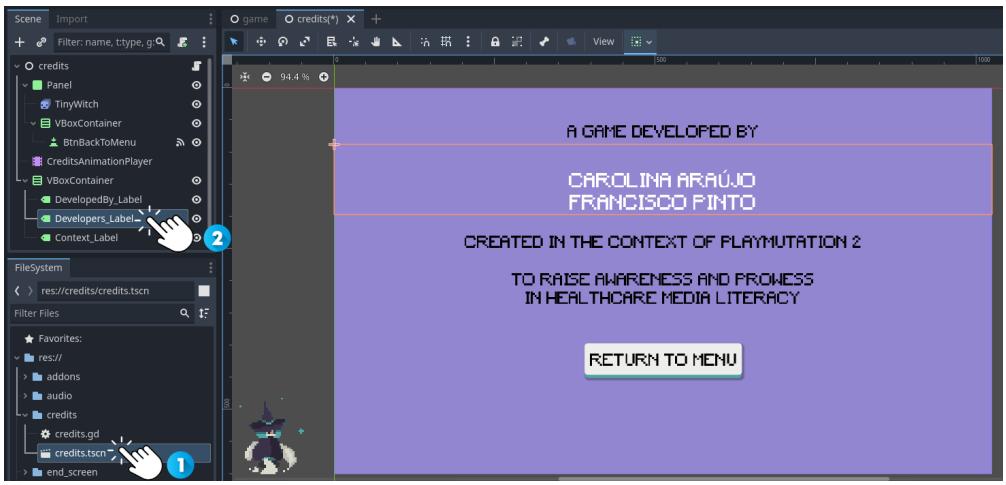

Figure 38: Click on the `Developers_Label` in the Credits scene.

2. On the Inspector section of Godot's interface (right side of the interface), edit the Text property of the `Developers_Label`. Press the **Enter** key to pass to a new line and write your name below the others (see Figure 39).

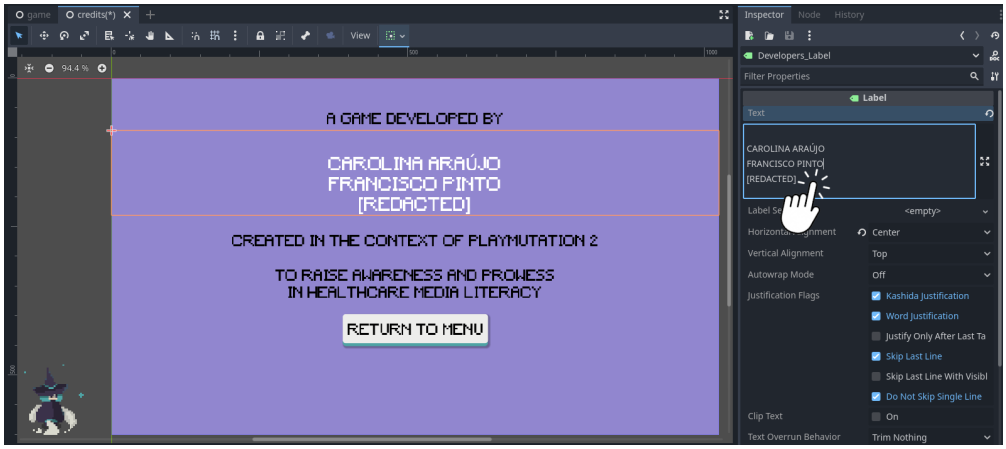

Figure 39: Add your name to the `Developers_Label` in the Credits scene.

3. Run the game again and, if you were successful in your attempt at making the Menu buttons work, check out the new, much-improved Credits scene!

3.2.9 I Will Take it to Go, Please

Fair, fair. All that is left to do to complete this mission is to pack up the game and export it. After exporting, you can save it, send it to others or publish it!

1. **Left-click** Project in the top-left corner of Godot's Interface and then **left-click** Export... (see Figure 40).

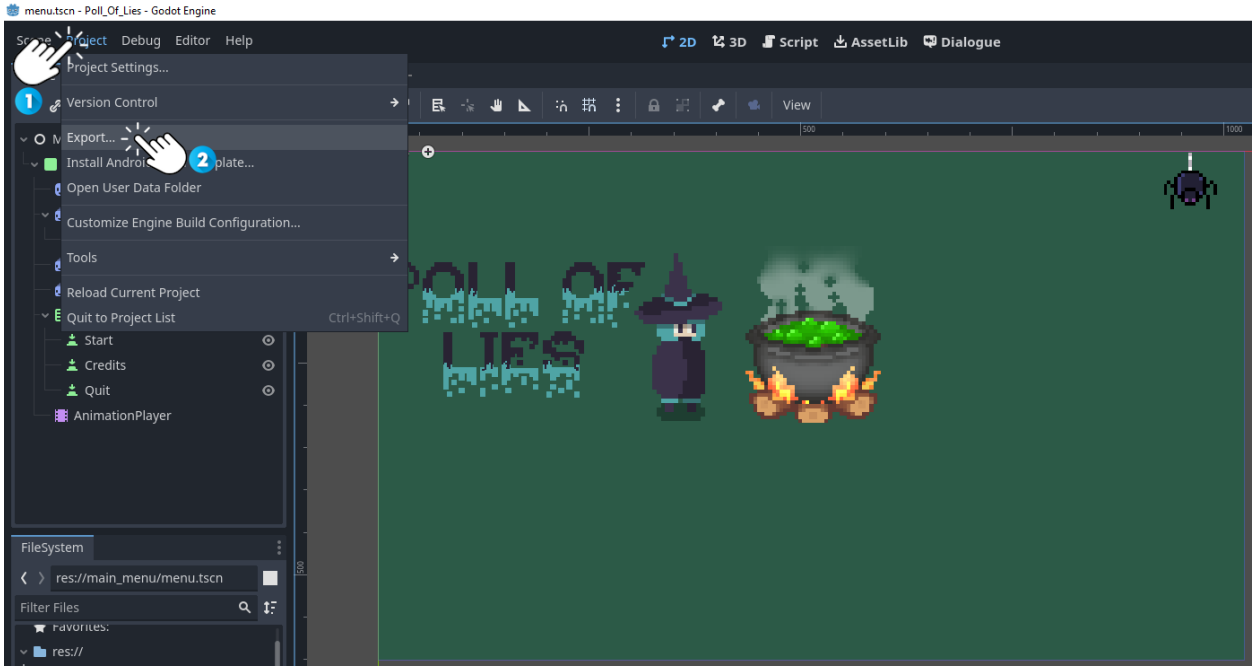

Figure 40: Click **Export...** to export your project.

2. A new window will open up. **Left-click** Add... and then, depending on the platform or operating system you want to export to, choose one of the available options. For demonstration purposes, we are going to show how to export to Windows Desktop, but the process for other platforms is similar from here on out (see Figure 41).

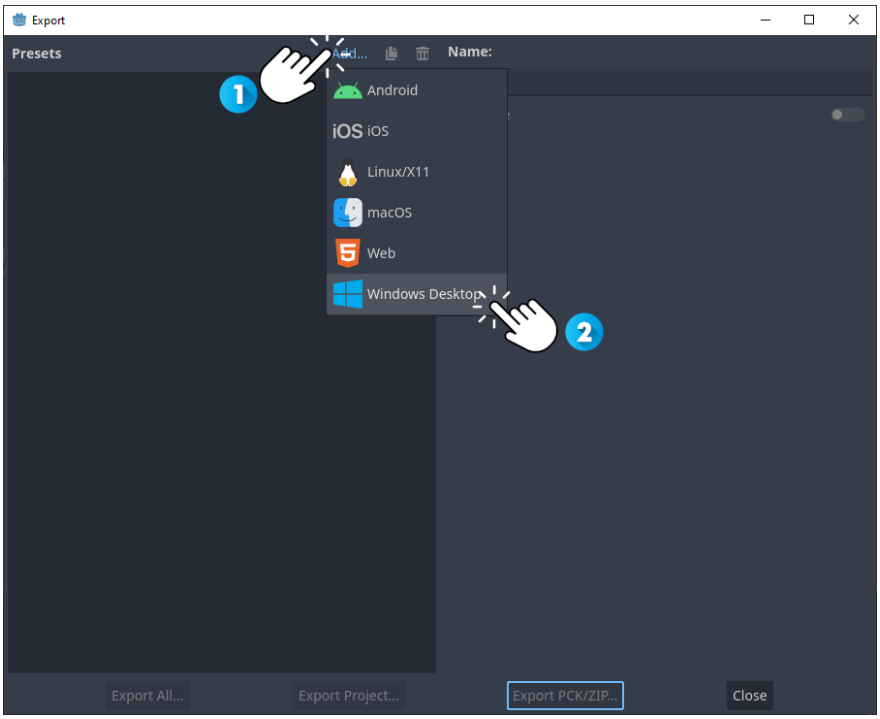

Figure 41: Choose the appropriate platform to export to.

3. **Left-click** Export Project... (see Figure 42).

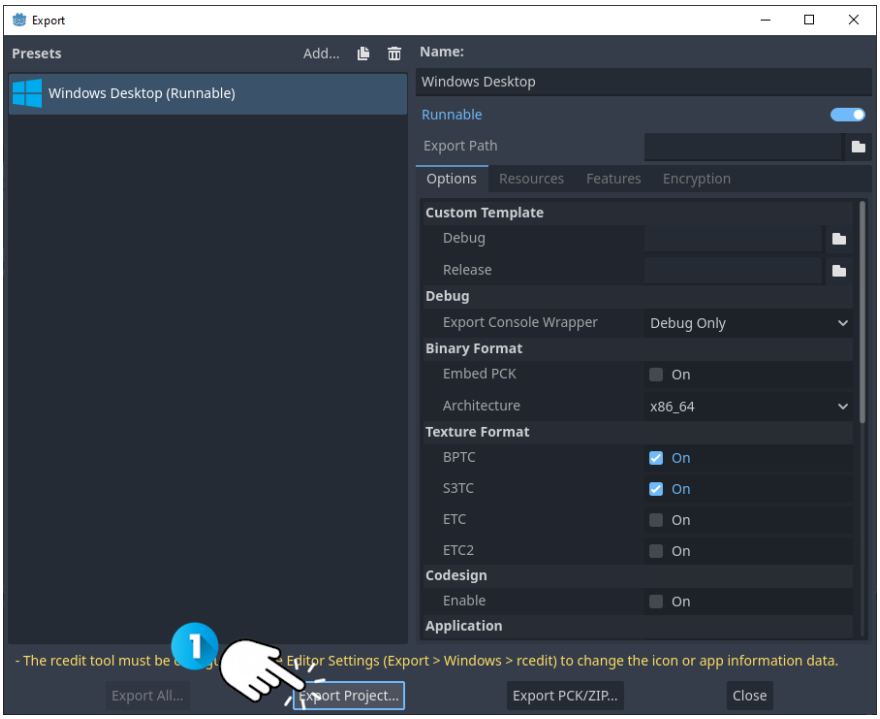

Figure 42: Click **Export Project**.

4. Search or create a folder in which you wish to save your game. When you have it, **double-left-click** the folder to go inside it. After this, you can **change the name** of your game. When you are satisfied, **left-click** Save and it is done! (See Figure 42).

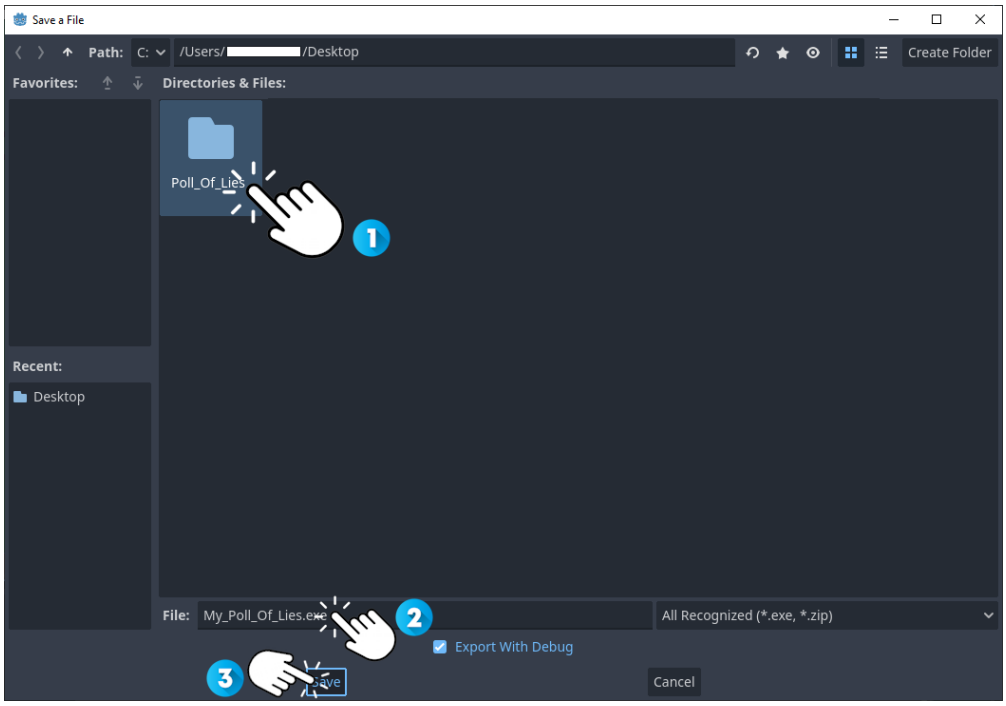

Figure 43: Choose the destiny folder and the name for the game and then click Save.

(Some warnings may arise, but you can ignore those, as all real programmers do (just kidding, sometimes they are important, but not in this case!))

5. The final version of the game is now on your computer, ready to be played and shared! Search for the folder in which you saved the game and open it. Inside, **double-left-click** the **.exe** file that **does not** have the word *console* in the name to play (see Figure 44).

   Enjoy **your** game. Remember the media literacy principles when choosing which post is true, and if you are unsure use the internet. Have fun!

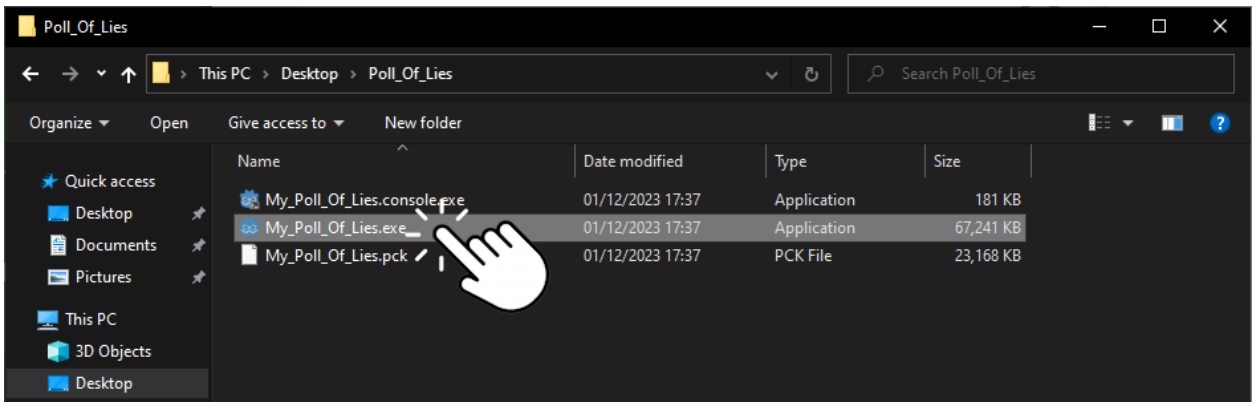

Figure 44: Locate the folder where you exported to and play your game.

3.3 Rewards

Current XP: 85

That was a big mission! Hopefully, you are now more prepared and confident to face new programming and game development-related challenges. This was a guided introduction to what goes on behind the scenes in games, but there is much more to learn and experiment with. Again, do not get discouraged if you could not understand everything and remember that there are tools available to help you understand everything that you are still unsure about.

3.4 Summary

Table 2: Documentation and Help - Where you can find answers to your questions.

Godot's updated documentation	Available at Godot Docs
Godot Engine Forum	Available at Godot Engine Q&A
Stack Overflow for anything related to programming	Available at Stack Overflow

Godot and programming concepts

- Godot is node-based. A scene is a node composed of other hierarchically structured nodes. Each node has a set of properties that you can change to best fit your game (visibility, text, size, color...).
- Godot uses GDScript, a programming language specific to this game engine. GDScript is indentation-sensitive, which means that sometimes tabs are necessary behind certain lines of code to create code blocks.
- To comment in GDScript, place a # behind the line of code.
- Nodes can have a script of code assigned to them, which is where the programming of behavior happens.
- Nodes can be connected through signals. These are useful to communicate that when something happens somewhere, something else should happen elsewhere. This connection can be made through the interface or code.
- A method or a function is a reusable block of code that executes when called. Methods can have arguments, which should be passed between parenthesis when called.

- All nodes have a `_ready()` method to control what happens at the beginning of the scene they are in.
- To go to another scene, you can use the method `get_tree().change_scene_to_file()`.
- To evaluate an expression and run code based on the result, use if-else conditions.
- Arrays are a type of data structure in programming where the items inside are organized by order of addition to the array. The first item has an index of 0, and it increases for subsequent items.
- Booleans are a type of variable that can either be true or false.
- Strings are a type of variable that holds text. Sum two phrases to join them in GDScript.
- Integers are a type of variable that holds whole numbers.
- All nodes have a `_input()` method that is called whenever an event happens, e.g. a mouse click.
- You can export your Godot project to different platforms, including Windows Desktop, Linux and macOS.

Misinformation

Misinformation in social media is a growing concern in today's digital landscape. With the rapid spread of information, the lines between fact and falsehood can become blurred.

The ease of sharing content on social platforms coupled with the potential for content to go viral amplifies the reach of misinformation, which can be anything from a false narrative, inaccurate data and misleading content. This phenomenon poses significant challenges because it not only undermines trust in information, but also influences public opinions and decision-making. Addressing misinformation requires a collective effort of users, involving media literacy education, fact-checking initiatives, and responsible content-sharing practices.

Be extra careful when dealing with information regarding healthcare to avoid adopting practices that do not fit your needs. Seek advice from a medical professional if and whenever possible.

4 But I Hate Programming

4.1 Mission description

Artists are important members of any team of game development, welcome aboard! The visual aspect of games is sometimes what matters the most, depending on the type of game, for example visual novels. This exercise serves as a way to explore your artistry and make the game more appealing (unless you can not even pick up a pencil, in which case, the game will become funnier).

4.2 Goals

4.2.1 My Drawings Can Move?!

As you may have seen by the tiny witch running around your game, animations in games are important to give it life and tell a story.

The way 2D games create animation is through a series of drawings of equally sized sides, or frames, played sequentially at a certain frame rate. To optimize the game and save as much space as possible, the frames of an animation are often placed together on what is called a sprite sheet (see Figure 45).

Figure 45: The different sprite sheets that compose the animations for the witch.

The witch you see in the game has six (6) different animations, all of which are composed of varying numbers of frames, combined in vertical sprite sheets. There are animations for the following actions: attack, charge, death, idle, run and receiving damage. This specific set of animations was obtained in Itch.io[8], a common place for indie game developers to share their projects, whether that is an actual game or art for games!

With complete artistic freedom, create your own frames! This mission is accompanied by a PNG file that you can draw directly onto with any type of software, e.g. Microsoft Paint, and create your sprite sheet. The sprite sheet template PNG can be found inside the folder **Level 2 - Poll of Lies → mission3**. Note that you do not have to draw on every single frame, do as many as you wish.

[8]Witches Pack by 9E0 available at https://9e0.itch.io/witches-pack

4.2.2 Stick Man is Valid

This project had no more budget for a real artist, such as yourself. To exemplify how to place your animation in the game, the following sprite sheet will be used, a true work of art nonetheless (see Figure 46).

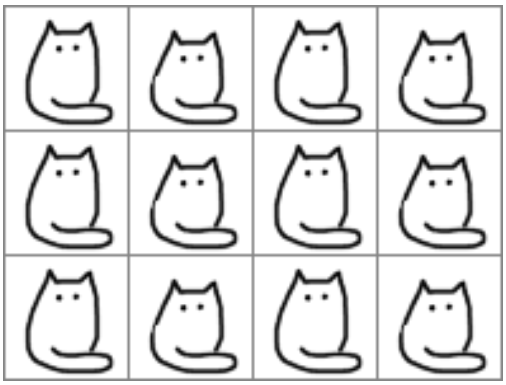

Figure 46: Tiny cat dance sprite sheet.

1. Save your drawing in PNG format. Open Poll of Lies in Godot.
2. Open the Menu in Godot's File System and search for the sprites folder. Place your sprite sheet inside this folder by **dragging** it from where you saved it (see Figure 47).

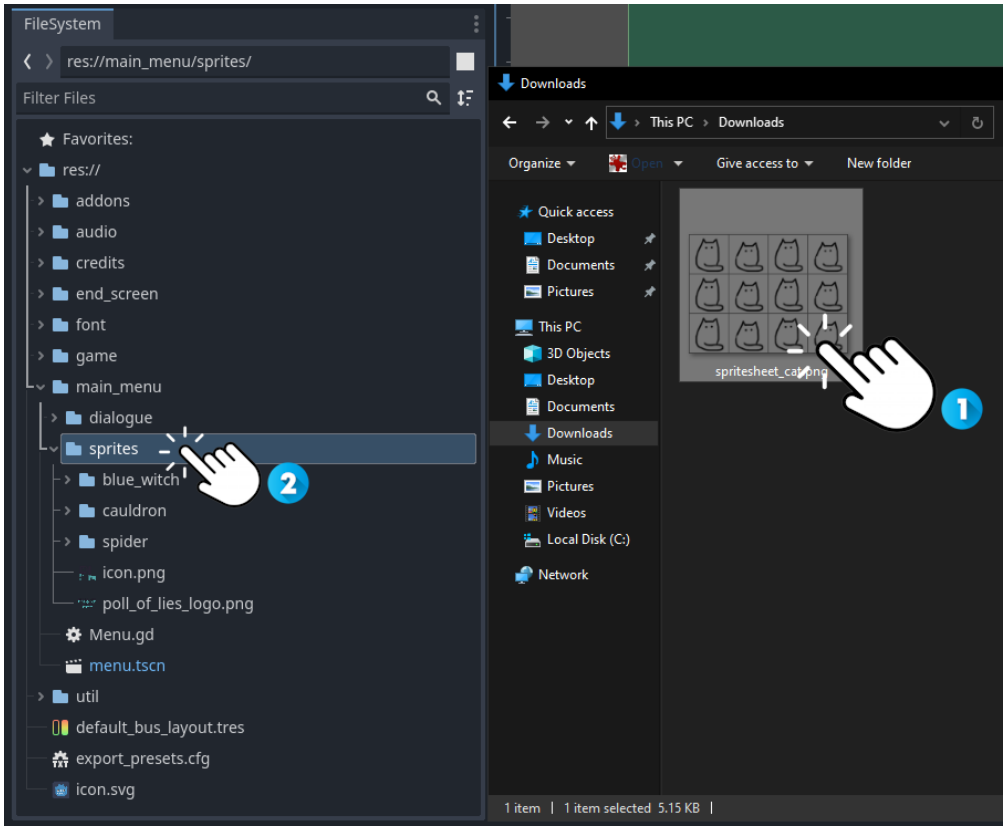

Figure 47: Drag your sprite sheet to the **sprites** folder of the game project.

3. **Right-click** on the Panel node of the Menu scene and then **left-click** on Add Child Node... (See Figure 48).

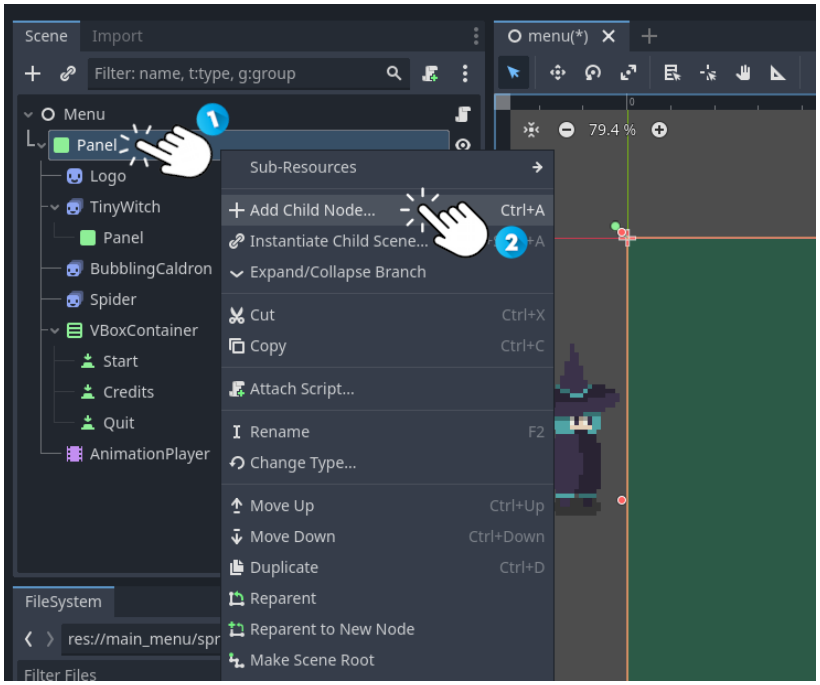

Figure 48: Add new child node to Panel in the Menu scene.

4. A new window will open up. At the top, click the search bar and write **animated**, to facilitate the search. When you find the AnimatedSprite2D node, **double-left-click** it to create a new one (see Figure 49). It will appear on the Menu scene's hierarchy.

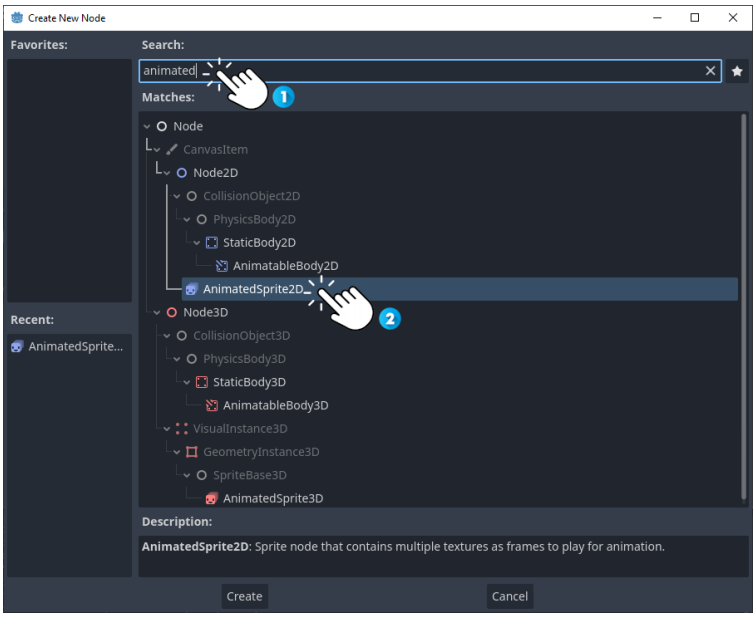

Figure 49: Create an AnimatedSprite2D.

5. **Right-click** the new AnimatedSprite2D node on the scene's hierarchy and then **left-click** in Rename. Change it to whatever fits best, but no curse words allowed! (See Figure 50)

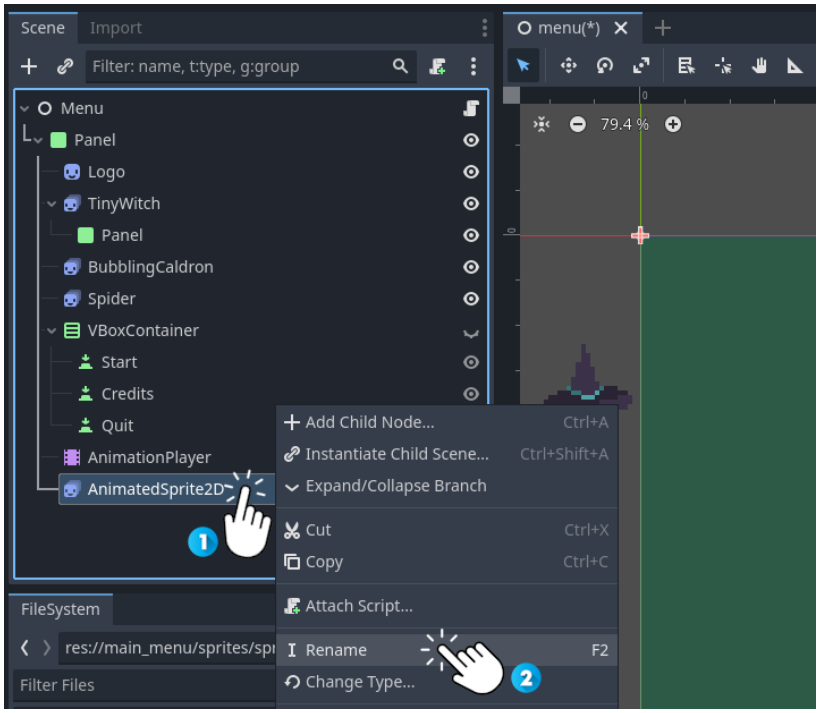

Figure 50: Rename your AnimatedSprite2D.

6. On the right side of the interface, on the Inspector, **left-click** the Animation dropdown and then **left-click** where it says <empty>. After, **left-click** on New SpriteFrames (see Figure 51).

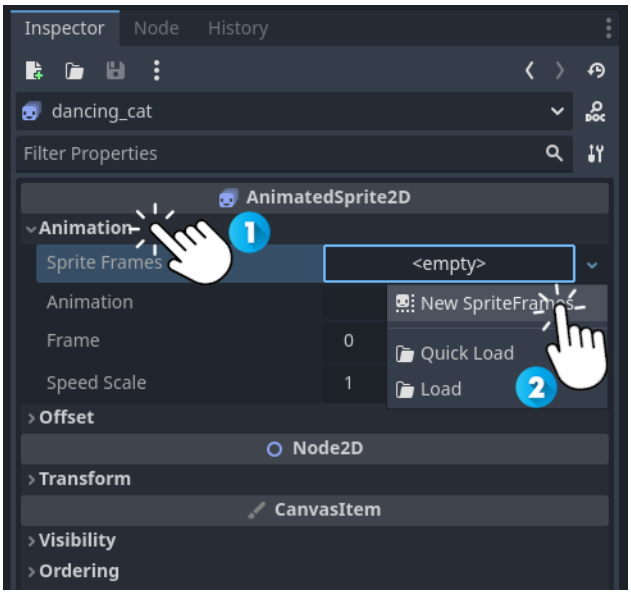

Figure 51: Add new SpriteFrames to the AnimatedSprite2D.

7. Still on the Inspector, **left-click** on the newly created SpriteFrames. You should be able to see a new area called **SpriteFrames** on the bottom panel (see Figure 52).

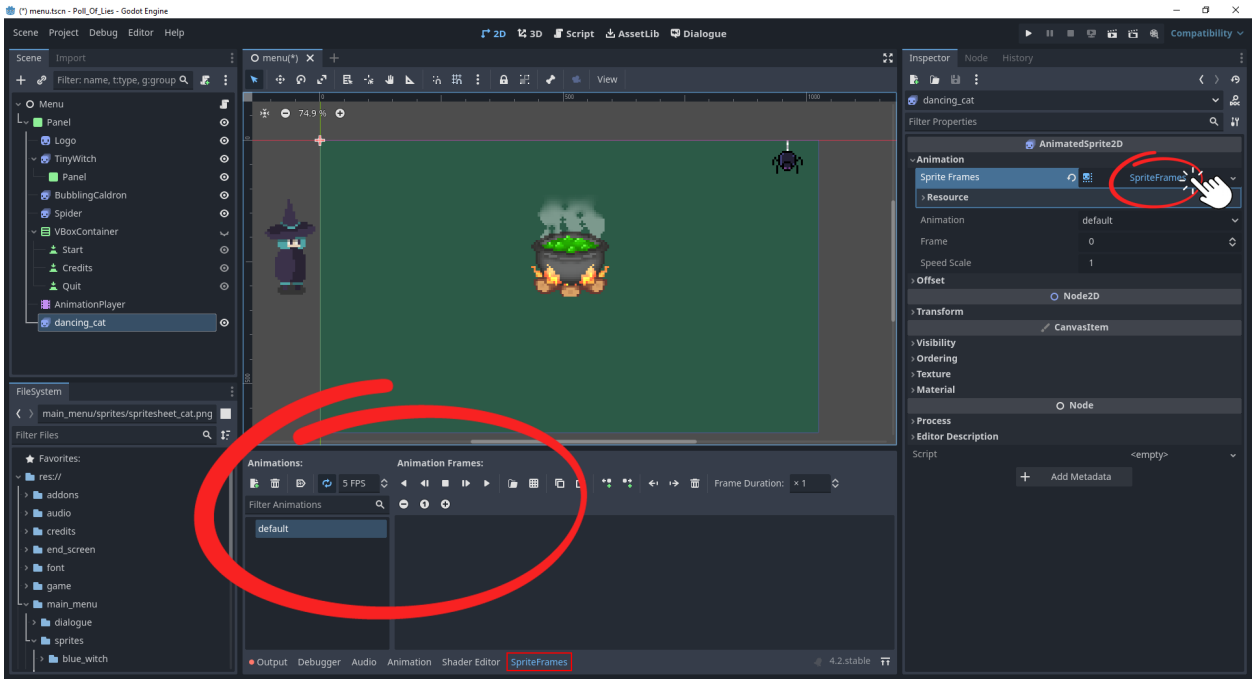

Figure 52: Open the **SpriteFrames** area of the bottom panel.

8. On the **SpriteFrames** of the bottom panel, **left-click** the small grid symbol (see Figure 53).

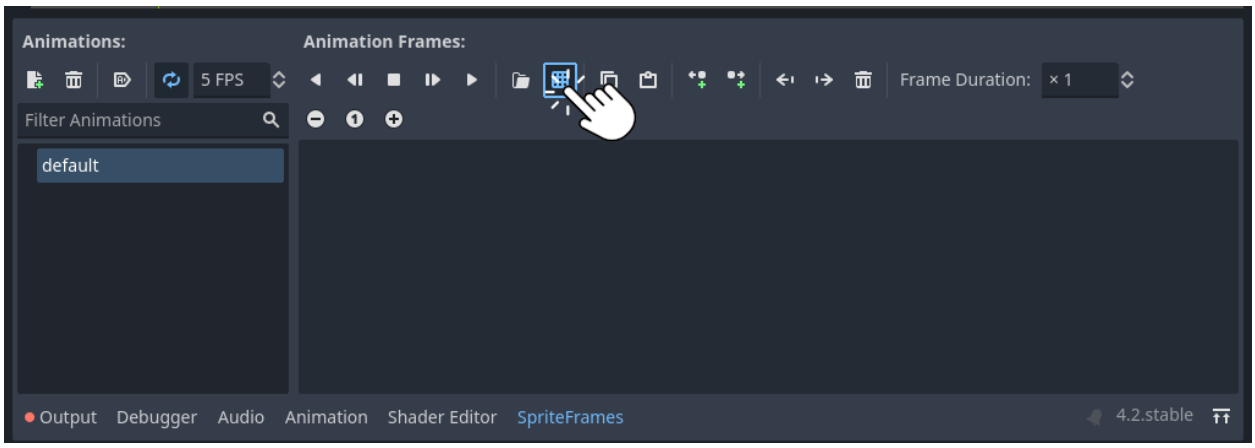

Figure 53: Click on the grid symbol of the **SpriteFrames** area of the bottom panel.

9. A new window will open and you will have to look for the **sprites** folder inside the **main_menu** folder, where you previously placed your sprite sheet. Click on your sprite sheet and then click Open (see Figure 54).

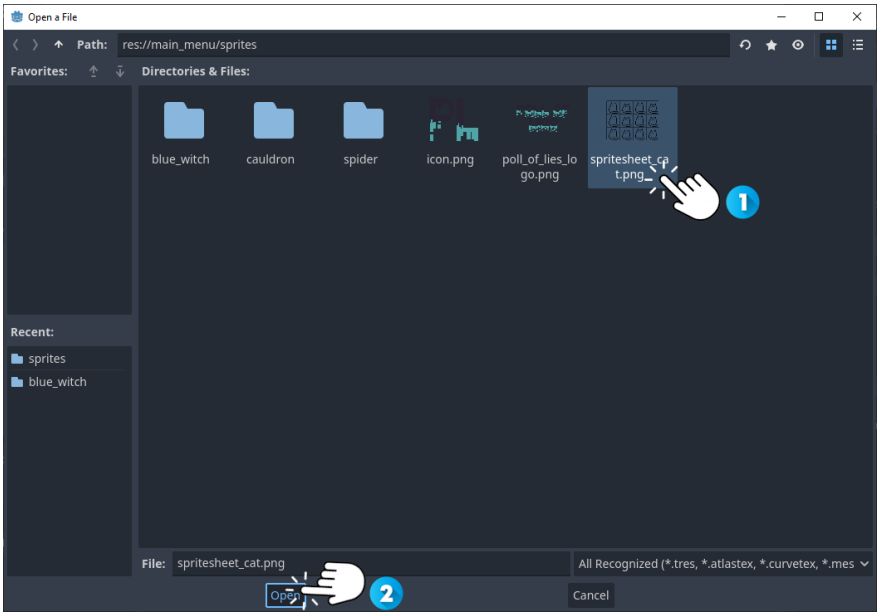

Figure 54: Open your sprite sheet.

10. A new window will open. Change the Vertical value to **3** (right side of the window). Then, sequentially, in the order you intend your animation to proceed, click every frame of the sprite sheet. If you did not draw on all the frames, leave out the empty ones. When you achieved the intended result, click Add x Frames. The window will close.

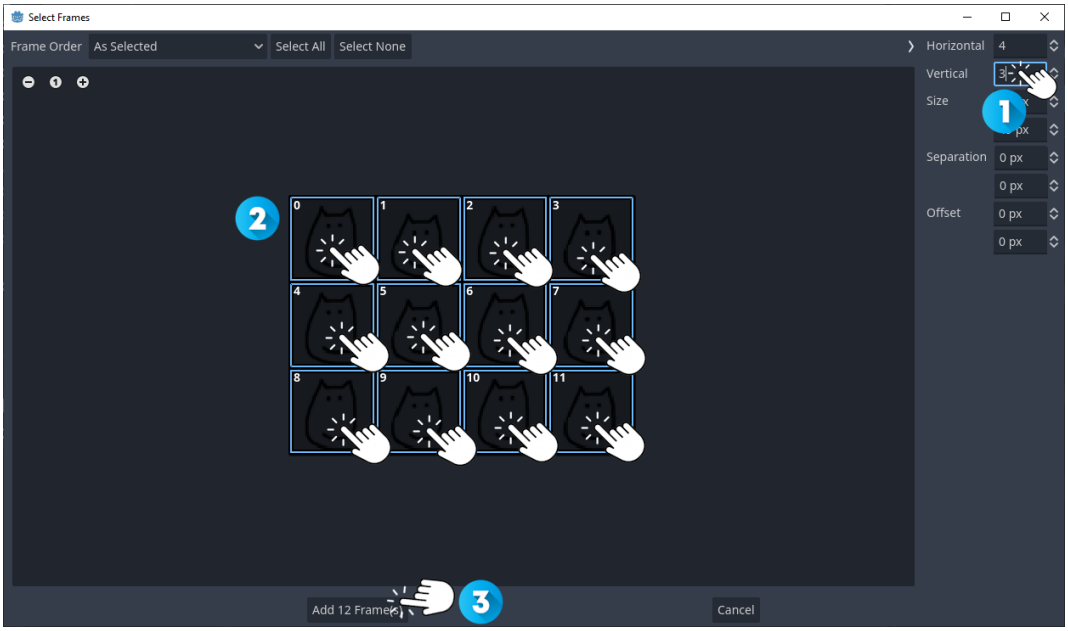

Figure 55: Select all the frames and add them.

11. Back on the **SpriteFrames** of the bottom panel, **left-click** the arrow symbol with an A for your animation to play automatically inside the game. If you wish for the animation to run faster, increase the value of the frames per second, or FPS, right next to the arrow symbol (see Figure 56).

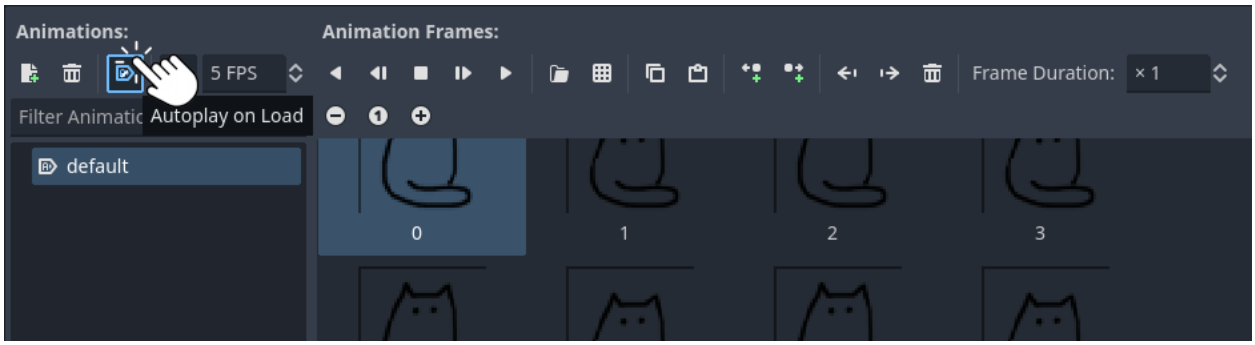

Figure 56: Autoplay the animation.

12. On the viewport of Godot's interface, find your animation within the scene and drag it to move it to whichever position you want (see Figure 57).

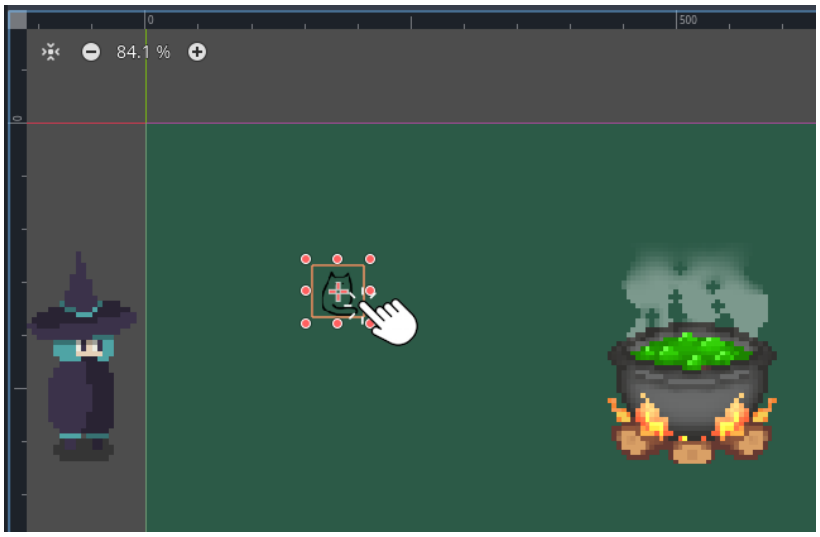

Figure 57: Move the AnimatedSprite2D inside the scene.

13. To scale up the animation, grab one of its corners and drag it outwards (see Figure 58).

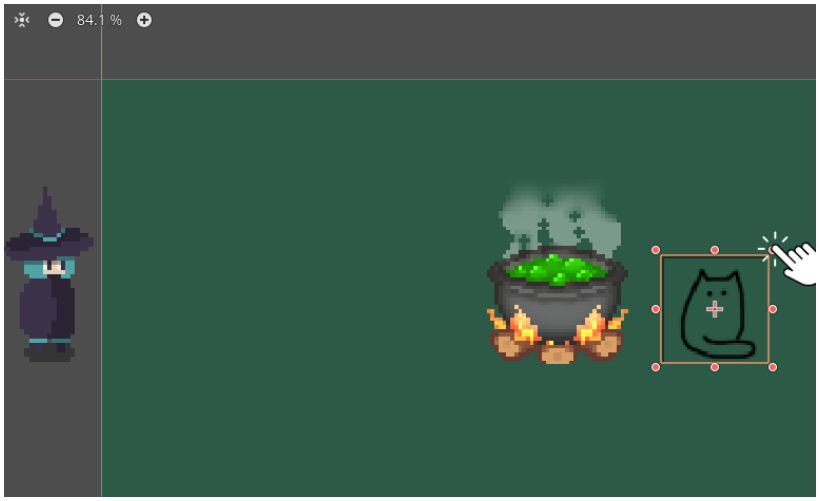

Figure 58: Scale up the AnimatedSprite2D inside the scene.

14. Run your game and check out your very own animation! The dancing cat turned out amazing, it really accentuates the vibe of the background music and ambience (see Figure 59).

Figure 59: Animation in the game.

If there are any black lines around your animation, worry not. Normally sprite sheets do not have lines indicating where each frame is, but it had to be done as such for the purpose of the exercise. You can remove the lines, draw your frames on a different layer on top of the template and then discard the template layer, or experiment with the Offset values when adding the frames for your AnimatedSprite2D (see Figure 55).

4.3 Rewards

Current XP: 100

Now that the game is adorned with your art, we shall steal it back and sell it for millions! Wait, no- the game was just appraised by famous game enthusiasts and **it is priceless**, all thanks to you! Here is your trophy, congratulations young game dev!

Figure 60: Best Poll of Lies Art Award.

Well played!
You have reached level 3.

References

[1] *SOLID Definition – the SOLID Principles of Object-Oriented Design Explained*, en, Apr. 2022. [Online]. Available: https://www.freecodecamp.org/news/solid-principles-single-responsibility-principle-explained/ (visited on 11/28/2023).

[2] *Warning Signs of Mental Illness*, en. [Online]. Available: https://www.psychiatry.org:443/patients-families/warning-signs-of-mental-illness (visited on 12/01/2023).

[3] *What is Mental Illness?* en. [Online]. Available: https://www.psychiatry.org:443/patients-families/what-is-mental-illness (visited on 12/01/2023).