1

OCR GCE A

COMPUTER SCIENCE PROJECT

H446-03

Name : Bryan Dissanayaka

Candidate Number : <INSERT CANDIDATE NUMBER>

<Institution Name> : <INSERT CENTRE NUMBER>

Title of Project : <INSERT PROJECT TITLE>

H446-03 – Project CONTENTS

Table of Contents

[A. Analysis 3](#_Toc83643749)

[Stakeholders 3](#_Toc83643750)

[PROBLEM 3](#_Toc83643751)

[RESEARCH 3](#_Toc83643752)

[SYSTEM REQUIREMENTS 6](#_Toc83643753)

[OUTLINE 6](#_Toc83643754)

[LIMITATIONS 7](#_Toc83643755)

[SUCCESS CRITERIA 8](#_Toc83643756)

[B. Design 9](#_Toc83643757)

[Systems diagram 9](#_Toc83643758)

[C. Developing the coded solution (“The development story”) 9](#_Toc83643759)

[D. Evaluation 9](#_Toc83643760)

[Project Appendixes 10](#_Toc83643761)

# A. Analysis

## Stakeholders

As the game is made to be easy to be played by anyone, I will give a demos of the game as I develop it to both developers and non-developers.

For developers, I can get feedback from friends and online game development communities which will be more objective and critical, and will comment on both the experience of the game and on the technical side of it. From the proposed solution, the developers will have a game they can analyse and gain knowledge from it. After I complete the project, I will be uploading the final coded solution to a public GitHub repository, along all the assets, so that they can download it and use my code in their projects.

For the non-developers I will ask friends and family which can describe whether or not the game is actually fun and engaging or not.

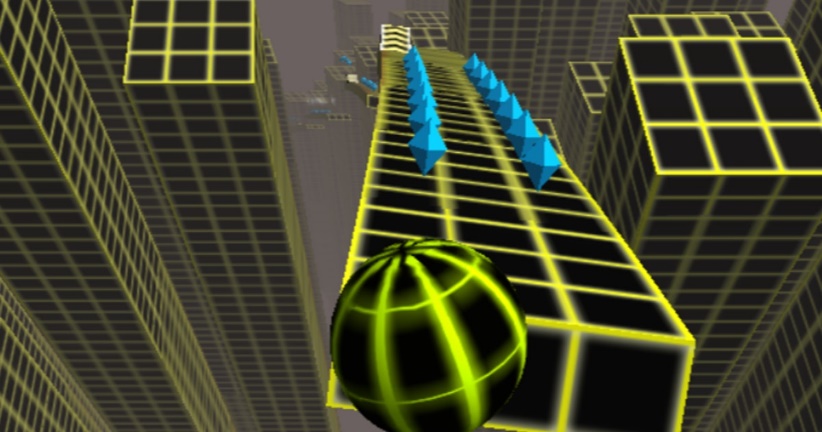
## PROBLEM

Most casual games like this do not give enough choices to the player in order to progress through a level in their own way. There is always only way to complete a level and everyone that plays the game will have the same experience and because of this.

To solve this problem, I will give the player a variety of things they can use to complete a level so that they can come up with their own unique strategy. This will also allow them to replay the same level but using a different strategy, therefore it will make the game automatically longer to be completely finished.

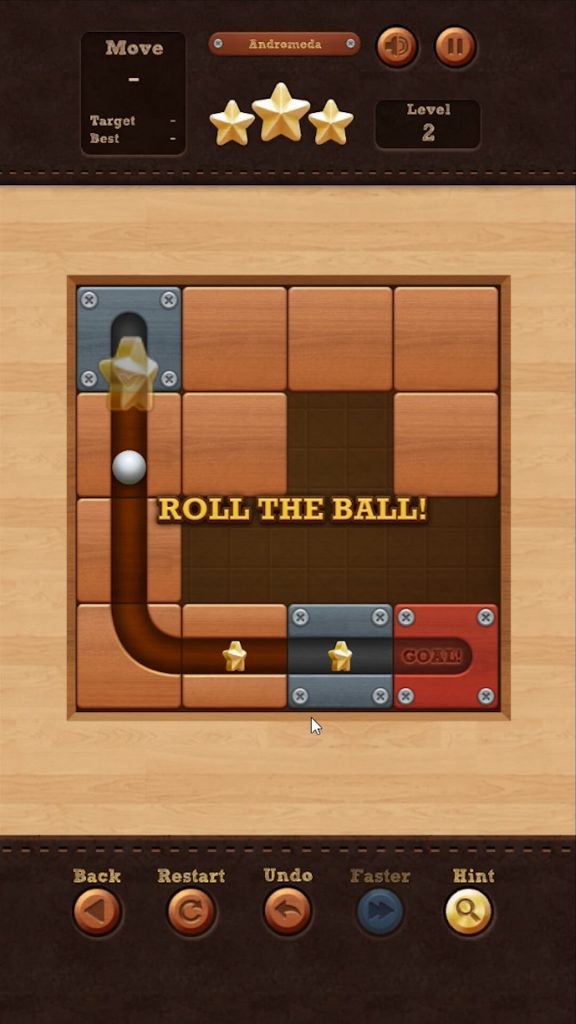
## RESEARCH

Plenty of roll-a-ball games have 3D graphics to impress the player and make them think that they will able to fully utilise the 3 directions, while only featuring 1 or 2 directions of input.



An example is a game called Crazy Roll 3D, where the player can only move the ball along 3 lanes and the ball automatically moves forward and the game stops when the balls collides with an obstacle. The input is 1 directional.

My game will have 2D graphics and the ball will only be moved by gravity acting downwards.



A game that comes closest to my game’s type of gameplay is Roll The Ball. The concept is similar: shape the level in order to complete it, with no control over the player, which is the ball.

My game differs from it as Roll The Ball features only one mechanic, which is to slide the tiles. As mentioned in the problems section, every single player playing this game will complete all the levels in the same way, and none of them have replay value because of this.

It is a fun casual game to play on a mobile device, but once the player completes all the levels, they will have no reason to keep playing the game.



An interesting mechanic my game will feature is inspired by a minigame from Super Mario 64 DS on the Nintendo DS. The players were able to draw trampolines on the touch screen in order to keep Mario from falling off. The angle of the trampoline mattered because to score points Mario would have to go through the rainbow coloured hoops. The size of the trampoline determined how bouncy they were. A smaller trampoline would make Mario bounce higher but it easier for him to miss it and fall down.

I want to create a similar mechanic where players will be able to draw slopes to make the ball roll in the direction they want.

## SYSTEM REQUIREMENTS

-Desktop

OS: Windows 10

CPU: Intel Core 2 Duo E8400 CPU

RAM: 4GB

GPU: NVIDIA GeForce 6200 or higher, compatible with OpenGL3

STORAGE: 1GB

-Mobile

OS: Android

RAM: 2GB

GPU: Compatible with OpenGL3

STORAGE: 1GB

\*\*These are the generic Godot system requirements and can change later

## OUTLINE

The game will feature a non-controllable player character. The player will move automatically and the user must

change the environment and objects of the level in order to get the player to the end goal.

The main inspiration where the game’s fundamental goal comes from is a minigame from Wii Party, where the players tilt their controllers to rotate the stage in order to roll a ball to the bottom of the level.



The main (and only) controls of this minigame are tilting the controller left and right. This minigame is also multiplayer, and the player reaching the end first wins. My game will not feature motion controls but mechanical and touch controls. To make the game more interesting I will add mechanics like enemies, lasers, different terrains that affect the ball’s speed, drawing shapes on the screen to create slopes etc. The player will be offered a variety of methods to complete each level of the game, and they will able to replay the same levels with less power-ups and mechanics to impose themselves with a more challenging experience.

The game will feature an infinite mode as well, where there is no end goal, but a high-score system where the score is based on how far down the player’s ball has travelled.

## LIMITATIONS

As the game is targeted to mobile devices, memory management and processing power need to be carefully assessed. The game will have 2D graphics as a 3D game with high polygon counts for the assets will consume the device’s battery much quicker.

For the infinite game mode, instead of instantiating new assets as the player goes down, the no longer used assets from above will be shifted down to reduce memory consumption. This creates a limitation in terms of game design as the game can seem to be repeating itself quite frequently.

Another method that could be used to solve this problem is to simply delete the unused assets when instantiating the new ones, so that memory consumption is constant. In this case the problem would be to load in the new assets required as the player moves down in a fast and seamless way. Sometimes it may cause stutters when large more complex assets are loaded in such as multiple enemy with specific behaviour and respective real-time pathfinding.

The game engine I am using, Godot, lets me select between OpenGL2 and OpenGL3. The first one will make the graphics of the game less demanding and it is the one recommended for porting the game as a web application (game will be run on a web browser). The second one will make the game look nicer and support higher resolutions, but it may cause performance issues if played on a web browser.

I will be using OpenGL3 as I did not intend to upload the game for web browsers. The advantage I would have by porting it as a web app would be that it would not require an installation for it to be ran. It would be easier to distribute as if I compiled the game to an .exe file format, most computers would recognize it as malware. This however is not a problem because I will publish the game on a website called itch.io so that it can be downloaded by anyone. It is a free platform for developers to share their game without paying a cost upfront (like Steam, which is also on desktop). All games uploaded to the website are scanned for potential malware that they can contain, therefore players feel safer to download games from it, rather than me distributing it myself (for example hosting a website for the game’s installation).

The required hardware is minimal for both desktop and mobile. Desktop will only need a mouse to play, but there will be keyboard shortcuts implemented. Mobile will only need touch controls. This is essential because as it is a casual game, users should not be required to plug in extra peripherals.

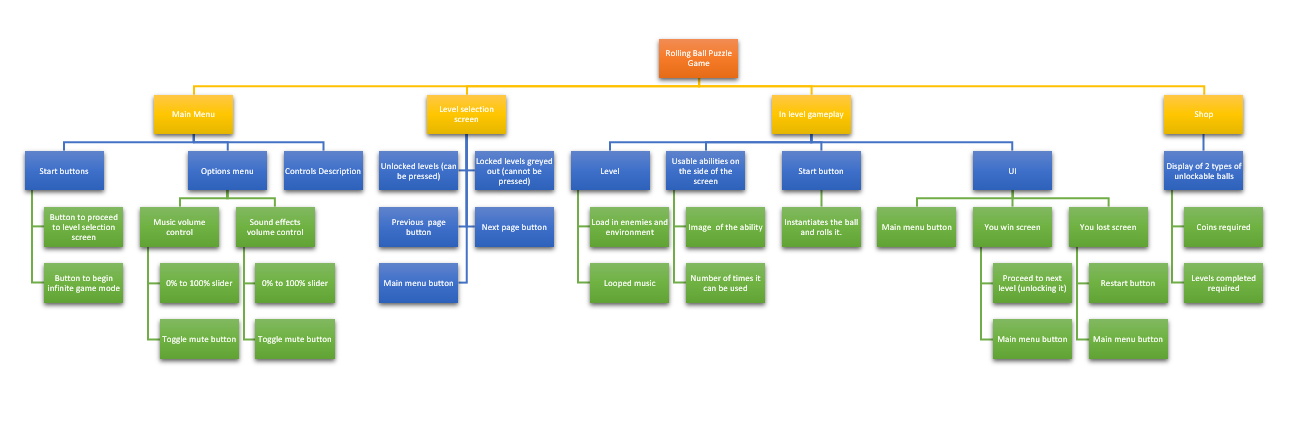
## SUCCESS CRITERIA

|  |  |
| --- | --- |
| **Requirement** | **Justification** |
| Vertical scrolling for infinite mode. | Since the ball is moved by gravity, this is the most intuitive perspective. |
| Drawing on the screen to create slopes. | Makes the player able to direct the ball in a certain direction or block a path. |
| Key to unlock doors. | The player cannot finish the level by going to the exit but must collect a key first. |
| Level is never bigger than the screen. | Player sees everything they need to know without needing to scroll to find the end of the level. |
| No controls require a keyboard. | Only a mouse is necessary to play the game. There will be keyboard shortcuts for adjusting settings for example, which can still be done using a mouse. |
| Mouse cannot go off the screen. | This is to not allow to draw slopes outside of the screen, which can possibly create bugs with the generation of next levels. |
| Responsive Menu UI | Seamless animations and transitions between level selection screens, options etc. |
| At least 10 different tools that the players can use. | These tools can be used by the players to complete the level. |
| Unlock tools progressively by completing levels. | Give the tools gradually to the players so that they don’t feel overwhelmed by having too many at the start. |
| Coins system | Collectable coins in the levels, |
| Shop system | Spend coins to buy upgrades or different balls. |
| 20 Levels | Enough for at least 1 hour of continuous gameplay. |

# B. Design

<See H446-03 Project Advice Booklet for help and guidance of what must go here.>

## Systems diagram



I made this flowchart to describe what the player’s experience would be when playing the game normally. This has helped me visualize the game’s development cycle better as I now know what the user should be presented with at every stage of the game. This chart however does not help me develop the game’s code so I decided to make another one to break down the code into smaller parts, so that I can develop an algorithm for every section.

## KEY VARIABLES

* **Game manager**

**Variables**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Example** | **When is it updated** |
| current\_scene | string | Holds the path of the current scene | “res://Scenes/Level1.tscn” | When changing scene |
| scenes\_dict | dictionary | Contains the paths of all the scenes in the game | scenes\_dict | Never because the number of scenes does not change at runtime. |
| coins | integer | Number of coins the player has collected | if(coin\_collected):  coins += 1 | When the player obtains a coin or when the player spends them in the shop. |

* **Music manger script**

**Variables**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Example** | **When is it updated** |
| menu\_music | string | Holds the path of the sound file corresponding to the menu music. | “res://Sounds/MenuMusic.wav” | Never |
| shop\_music | string | Holds the path of the sound file corresponding to the shop music. | “res://Sounds/MenuMusic.wav” | Never |
| gameplay\_music | string | Holds the path of the sound file corresponding to the gameplay music. | “res://Sounds/GameMusic.wav” | Never |

To play the music, I will create 3 Nodes in Godot of type AudioStream for each soundtrack, and in each of their scripts, I will pass in the string of the corresponding file

* **Player variables**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Type** | **Description** | **Example** | **When is it updated** |
| velocity | vector2  [This is a data type in Godot that holds 2 floats. The first one is accessed with vector2.x and the second with vector2.y] | The horizontal and vertical components of the player’s velocity. | // if a tool can modify the ball’s velocity:  velocity.x \*= 1.5 | Every frame as the ball is constantly accelerating or decelerating depending on the environment (slopes, collisions, enemies, etc.). |
| ball\_sprite | Sprite Node  [Godot component responsible to handle the sprite of an object] | Holds the sprite component. | ball\_sprite = get\_node(“ball\_sprite\_node”)  ball\_sprite.set\_texture(redSprite) | When the set\_texture function is called,  so when the player wants to change the look of the ball. |
| is\_frozen | bool | If true, ball will be frozen in place | //When set to false, ball starts moving  if(start\_pressed):  is\_frozen = false` | When the player presses the start button the ball starts rolling, so it will be set to false. |

## USER INTERFACE DESIGn

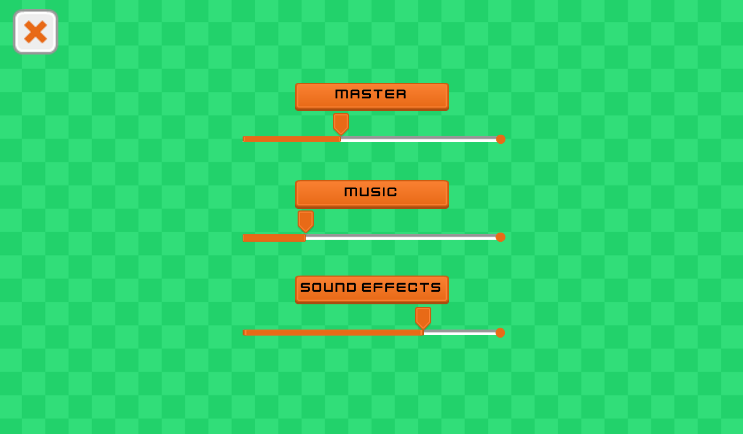
The assets I am using for the UI are from an asset pack. They are completely free for all purposes and they can be downloaded from here: <https://www.kenney.nl/assets/ui-pack> .

### main menu

The menu screen is minimalistic and self-explanatory. The start button will transition the screen to the level selection screen and the options button will transition to the options menu. The X on the top left of the screen will close the game.

I will have animations of balls rolling down the screen as the background of this page.

### Option menu

The option screen will feature sliders to control the sound of the game. There will be three for master, music and sound effect. The X in the top left in this screen will transition to the main menu above instead of closing the game entirely.

I decided not to include a resolution selector as I would have to remove it for mobile releases, therefore making two different versions of the game: one with the selector for desktop and one without for mobile. I believe that it is not necessary anyway because it will be running in a locked ratio of 16:9, resizable window on desktop.

### level selection screen

In the level selection screen, each level will be colour coded and in some cases a symbol under it.

Levels that have been completed will be yellow, and each level will feature an optional challenge that if completed, will grant the player extra coins. Levels where the challenge has been completed will have a checkmark under their number.

Uncompleted levels are grey and if they are locked they have a X under their number.

### gameplay interface

This is the layout of the UI in a level.

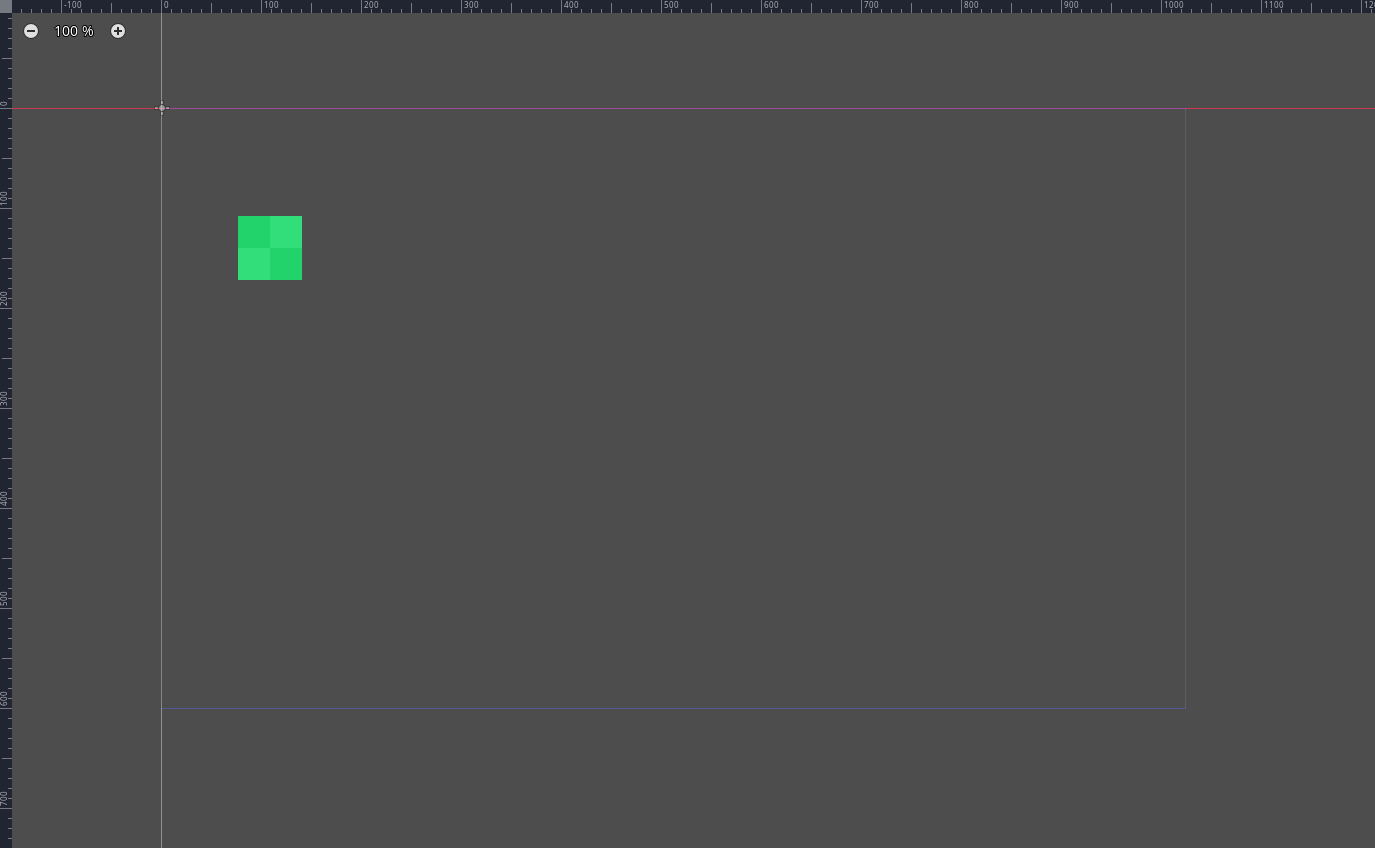
The green side is where the actual level is presented (ball, slopes, enemies, walls etc.). The right grey panel is where the available power ups and tools are.

The player would select a power up (blue squares that will have icons for each power up) and then use it. There is also a slider on the right to scroll up and down the list.

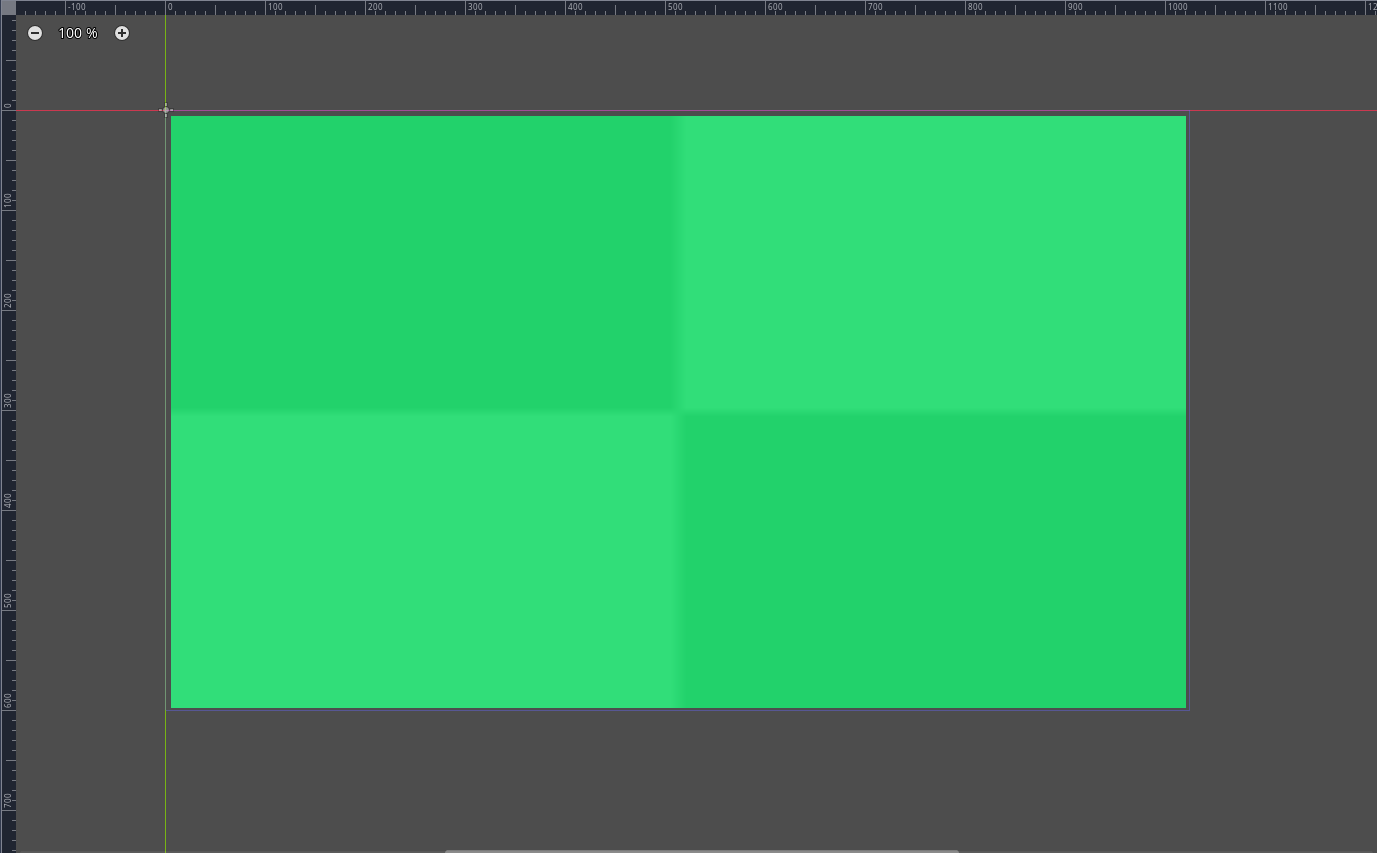
The green checkmark is pressed to start rolling the ball. The red cross on the top left is for exiting the level. I have made it smaller so it is harder to accidentally press during gameplay. When pressed it will prompt the user if they want to leave the level, losing their progress in it, for an additional level of security against accidental presses.

To create the green background, I used this image. The image is from the rolling ball asset pack, which I also downloaded from Kenney: <https://www.kenney.nl/assets/rolling-ball-assets> .

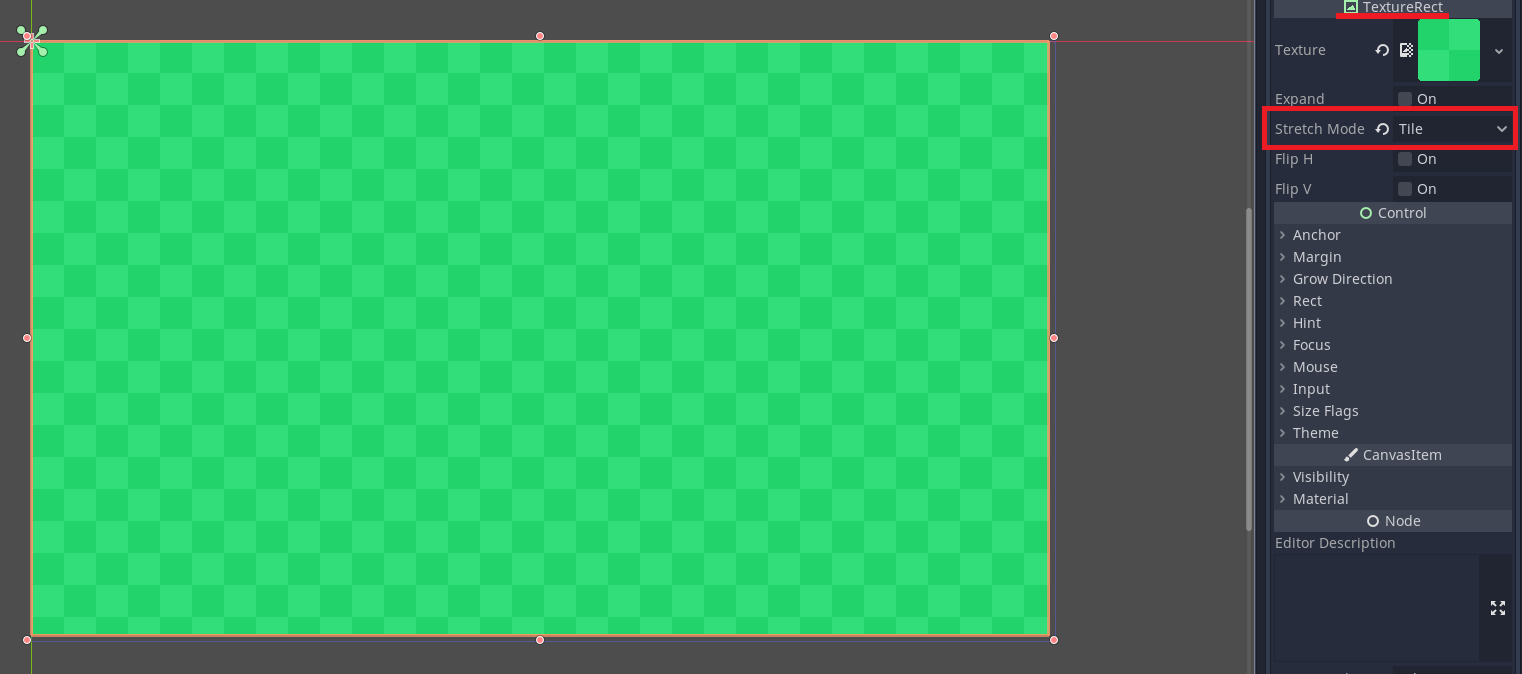
To create an interesting look, I decided to repeat the tile across the background, without needing to duplicate it and align it multiple times. I decided to import it into Godot and check the documentation online to see if there was a feature to tile an image when resizing it instead of stretching it.



This is how it looks like when it is simply imported with the default settings.



This is what happens when the image is resized to fit the background size (which is the size of the game window).



By creating a node of type TextureRect, assigning its texture to the image and setting the stretch mode to tile, I got the image to tile correctly across the entire windows size. Then I used this tiled image in PowerPoint to create the concept art for the user interface.

###### stakeholders’ inputs

These GUIs will be tested by the stakeholders and are subject to change based on their feedback. For now, I think the general layout of the UI is very common so the players should not be confused when first launching up the game.

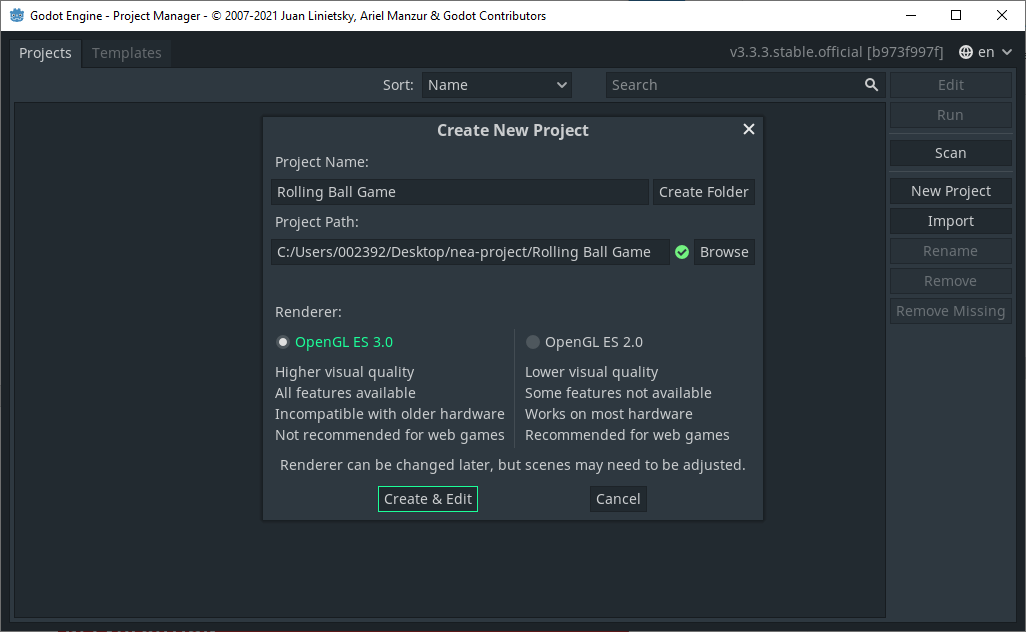
# C. Developing the coded solution (“The development story”)

## SETTING UP THE PROJECT

##### Git repository

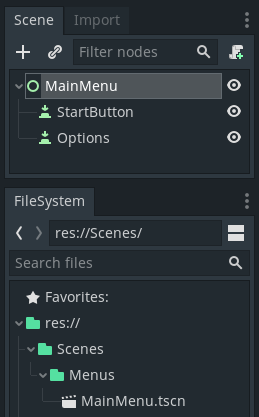
I am working on the project both in school and at home, which means I am constantly switching devices. Because of this, I decided to use Git version control, instead of saving the project files on a USB. Git is a version control system, which means it records changes to a set files so that I can recall specific version of those files later. I already used Git in the past for personal projects which is why I decided to use it instead of other version control system such as Plastic SCM or Helix Core. Using version control also makes sure that my project is backed up.

##### CREATING THE PROJECT



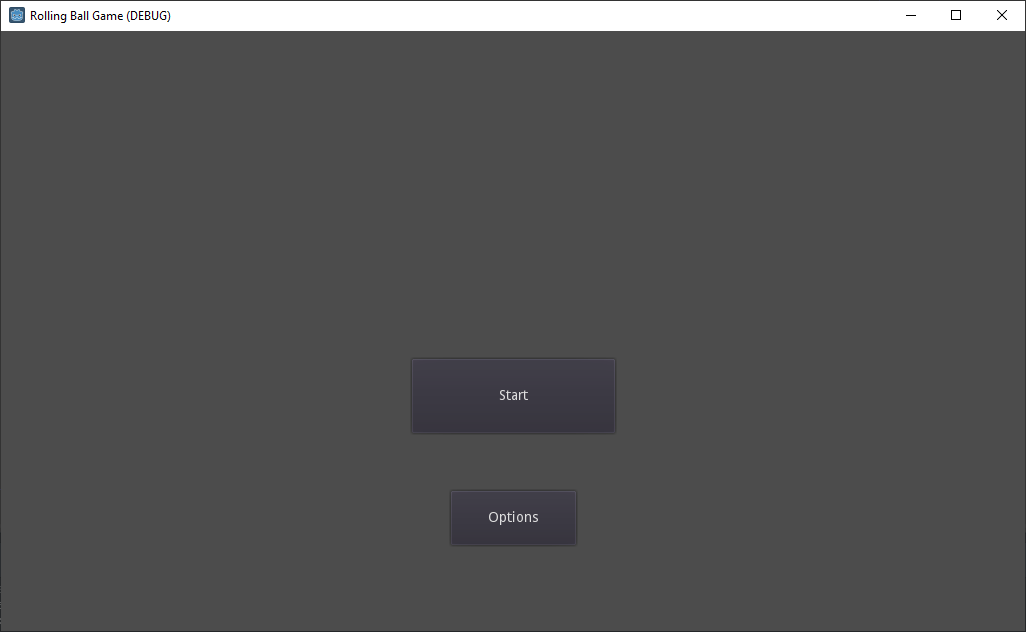
I created an empty folder in the Git repository folder, and selected it as the project path. I am using OpenGL 3 as I mentioned in the design section because I will not export this game as a web game.

## MAIN MENU

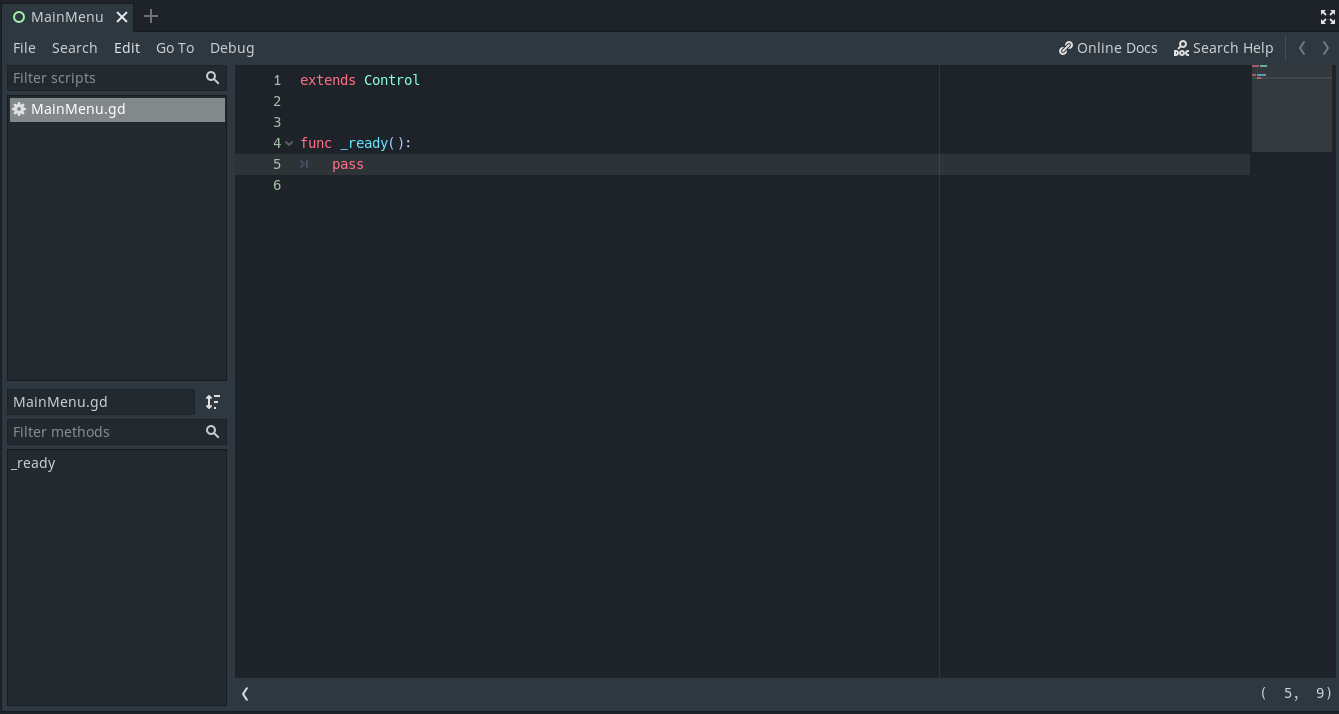
The first I am developing is a simple main menu.

For the master node, I created a node of type Control, which is the base Node for UI-related nodes and named it MainMenu. I created 2 buttons: 1 for the Start button and one for the Options button.

I created a folder called Scenes and a subfolder called Menus where I will store all the menu scenes.



This is how the game looks like when it is launched. The buttons show up but when pressed, nothing happens, so I will implement their functions now.

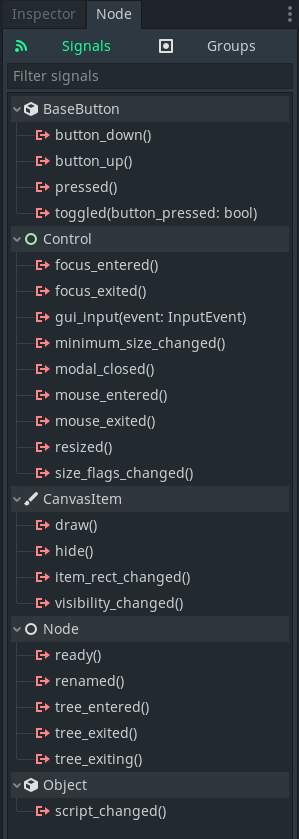


I created a script called MainMenu.gd and attached it to the MainMenu master node.

The extends keyword is to represent class inheritance. When creating a script for a node, the script acts as a subclass of that node. Since the MainMenu node is a Control type node, the script is a subclass of Control.

The \_ready() function is called when the object is instantiated. The pass keyword ends the function.

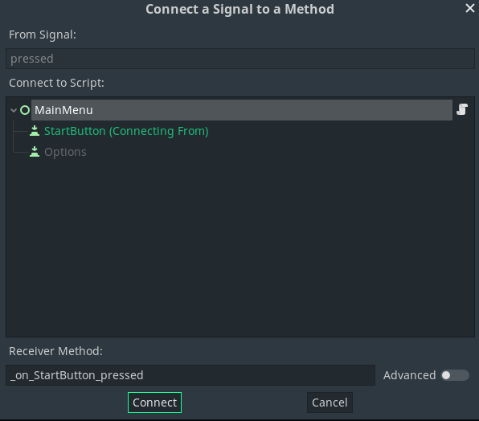
This script will contain the methods of the actual functions of each button. The methods are called when the buttons are pressed so I will use Godot’s signal system to call functions across nodes.

Each node has a list of Signals it can send.

From the Godot official documentation: “*They allow a node to send out a message that other nodes can listen for and respond to. For example, rather than continuously checking a button to see if it's being pressed, the button can emit a signal when it's pressed.”*

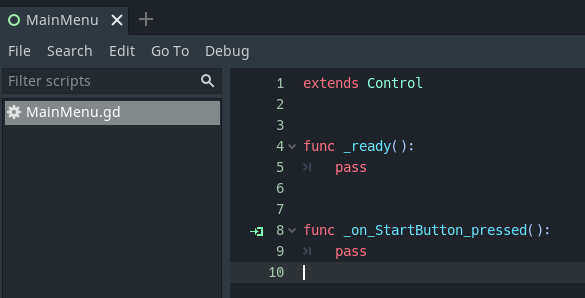
Godot is object oriented, so the list of signals follows a class diagram order. StartButton is a Node of type BaseButton, so it inherits from the BaseButton class, which inherits from the Control class. The control class (or node type) inherits from CanvasItem on so on. Therefore, the node StartButton will be able to these signals.

For sending a signal when it is pressed, I will use the pressed() signal.

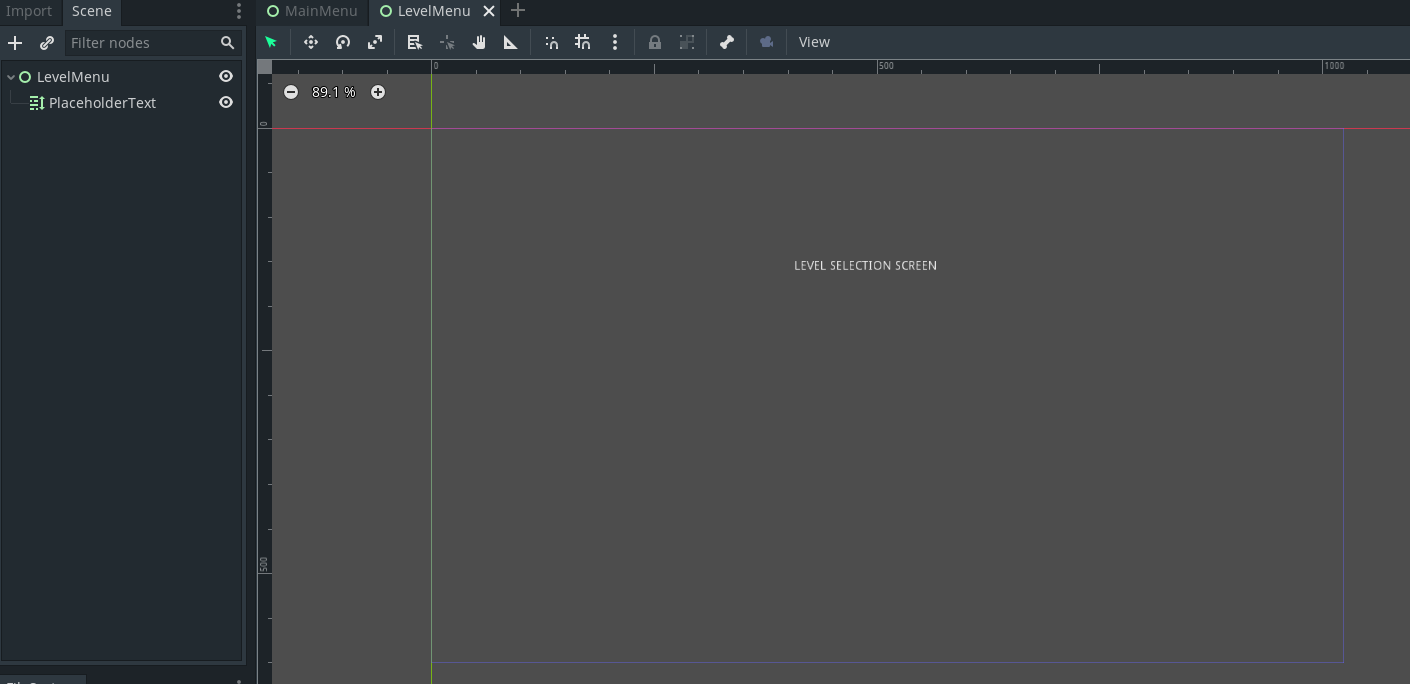


Godot provides a GUI to connect a signal to a script.

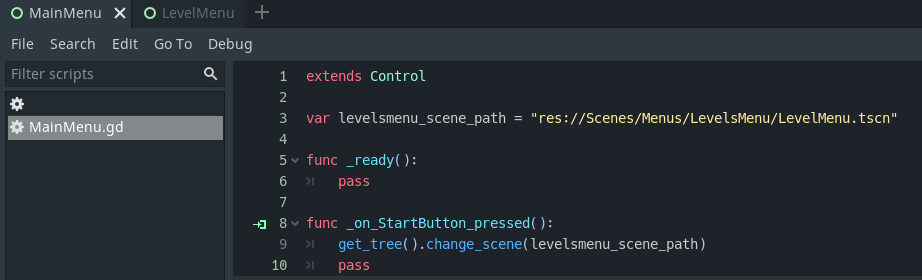
In the Advanced section I can choose extra call arguments, but for now I believe this method does not need any.

Now the signal is connected to the MainMenu.gd script. When the button is pressed, the pressed() signal will call the \_on\_StartButton\_pressed() function in the MainMenu.gd script. I can now write the behaviour of the method in the function.

This is the StartButton which will transition the game into the Level selection screen. So I will create a basic scene for that first.



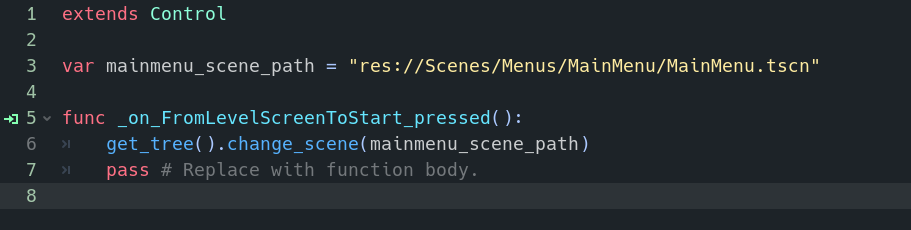
Since I am developing the main menu first, this level selection screen contains only a main node of type Control (like in the main menu) and a Text Node acting as a place holder.



In the MainMenu.gd script, I created a variable to hold the path of the LevelsMenu scene. When the start button is pressed, \_on\_StartButton\_pressed() is called, which gets the root of the scene tree and calls the change\_scene() function. It passes levelsmenu\_scene\_path as the argument.

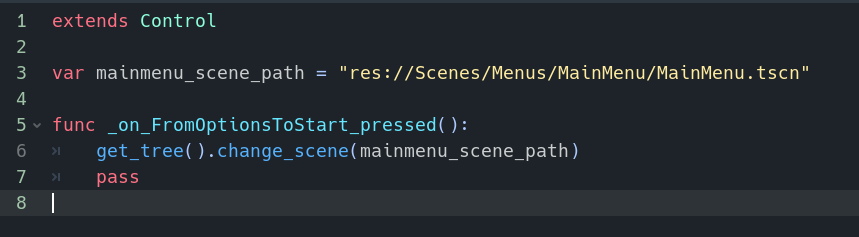
Now when pressing the start button, the game transitions into the Level Selection Screen (refer to Video\_01.mp4).

I repeated this process to create a button in the Level Selection Screen that when pressed, it transitions the game back to the Main Menu (refer to Video\_02.mp4).

Here is the script LevelMenu.gd attached to the master node of the LevelMenu scene: 

I used the signals system of Godot exactly in the way I used it to make the start button.

I created a new scene for the Options Menu called OptionsMenu.tscn and designed it just like the Levels screen and added a button to go back to the main menu (and placeholder text). (Refer to Video\_03.mp4)



* **Testing of current UI**

|  |  |  |  |
| --- | --- | --- | --- |
| **Button** | **Expected result when pressed** | **Tested Result** | **Video** |
| StartButton | Transition to level screen from start menu. | Transitioned to level screen. | Video\_01.mp4 |
| FromLevelScreenToStart | Transition from level screen to main menu. | Transitioned from level screen to main menu. | Video\_02.mp4 |
| OptionButton | Transition to option screen from menu. | Transitioned from start menu to option screen. | Video\_03.mp4 |
| FromOptionsToStart | Transitions from option screen back to the main menu. | Transitioned from options screen to the main menu. | Video\_03.mp4 |

# D. Evaluation

<See H446-03 Project Advice Booklet for help and guidance of what must go here.>

# Project Appendixes

Insert as many project appendixes as you need for your project.

These might include, but are not limited to:

* Complete Code Listing (ESSENTIAL)
* Interview Transcripts
* Meeting notes
* Observation notes or questionnaires