-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.

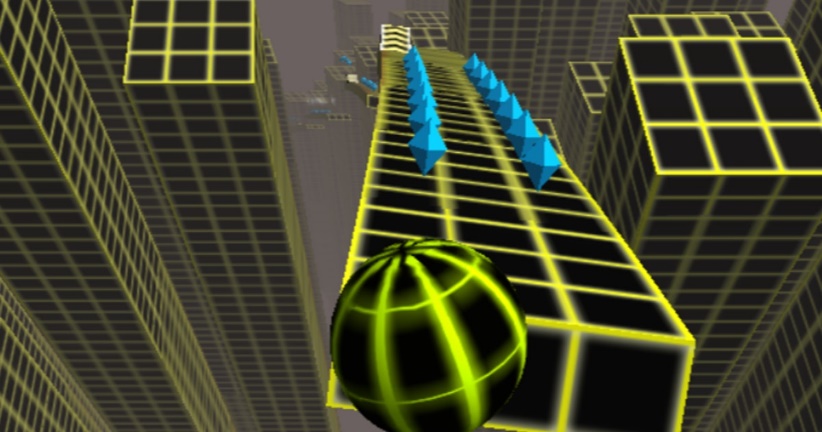
## PROBLEM

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.

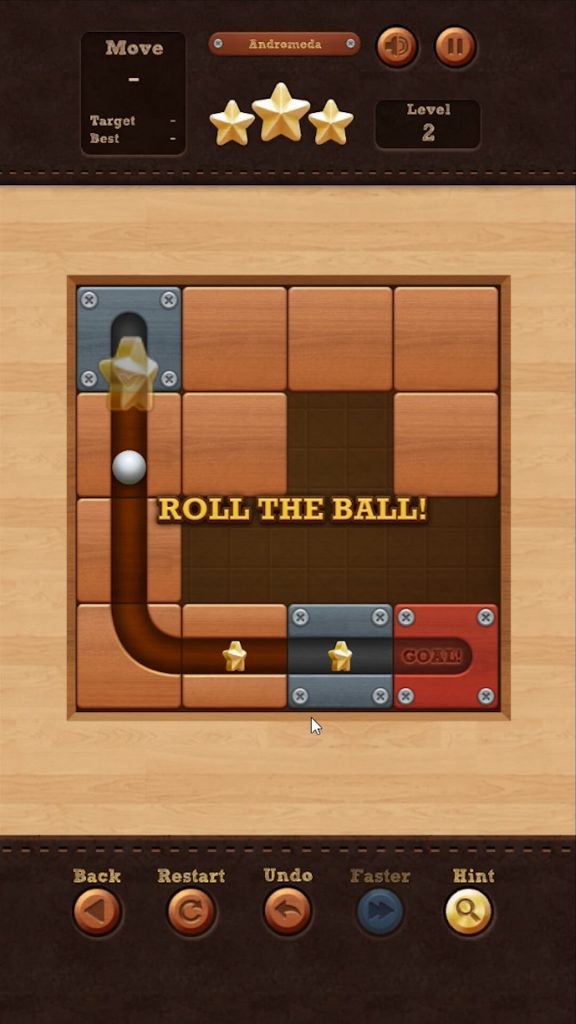
## RESEARCH

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 on the screen 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 enemies with specific behaviour and respective real-time pathfinding.

The advantage I would have by porting the game 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).

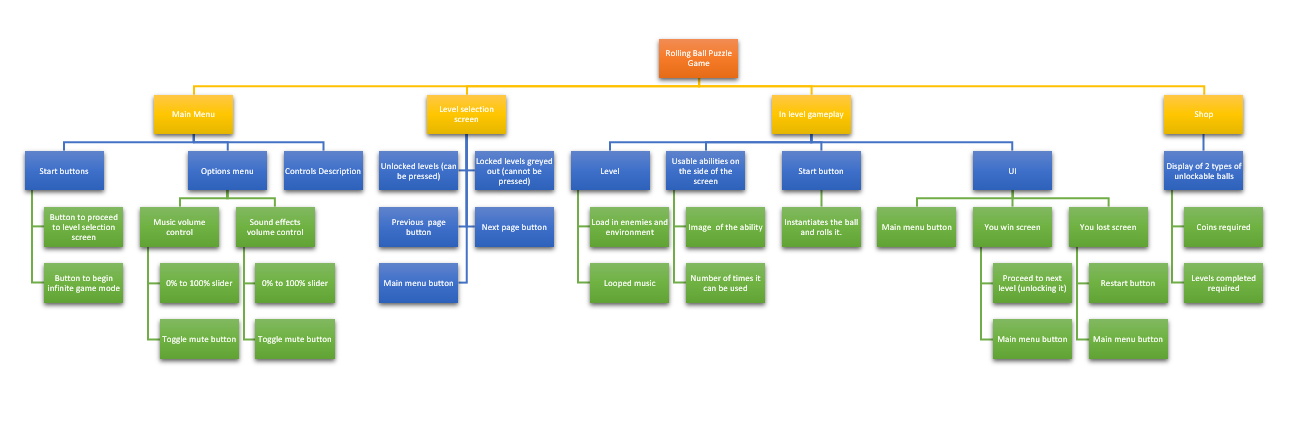
## SUCCESS CRITERIA

|  |  |
| --- | --- |
| **Requirement** | **Justification** |
| 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. |
| 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 | Stutter-free transitions between level selection screens, options etc. |
| At least 5 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. |
| Stars system | Collectable stars in the levels (3 in each level) |
| Shop system | Use earned stars to obtain different balls. |
| At least 10 Levels | Enough for at least 30 minutes of gameplay. |
| Ghost enemy | An enemy that will be able to follow the player ignoring the collisions in the level and kill the player on contact. |
| Patrolling enemy | This enemy will have an array of predefined positions in a level. It will travel along straight lines between those positions, and staying in them for a set amount of time before travelling to the next one. |
| Teleport tool | Player will be able to teleport in a direction of their choosing. The maximum teleporting distance will be able to be changed for each level. |
| Gravity tool | The player will be able to revert gravity with the press of a button. |
| Freeze tool | Ability to completely stop the player’s ball (and unfreeze it). |
| Push tool | Apply instantaneous forces to the player’s ball in a chosen direction in order to launch it left or right. |
| Saving and loading system | The player’s progress will be saved so that when closing the game and opening it again, the progress is maintained. |

# 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.

## USER INTERFACE Design

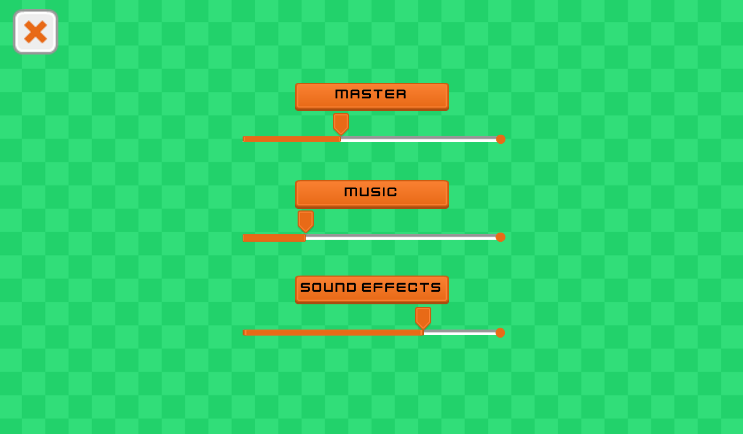
The assets I am using for the UI are from an asset pack. They are completely free for all purposes and they can be downloaded from here: <https://www.kenney.nl/assets/ui-pack> .

### 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 plan on animating the background as well.

### 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. 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 file 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, in case I lose the project files or if they get corrupted.

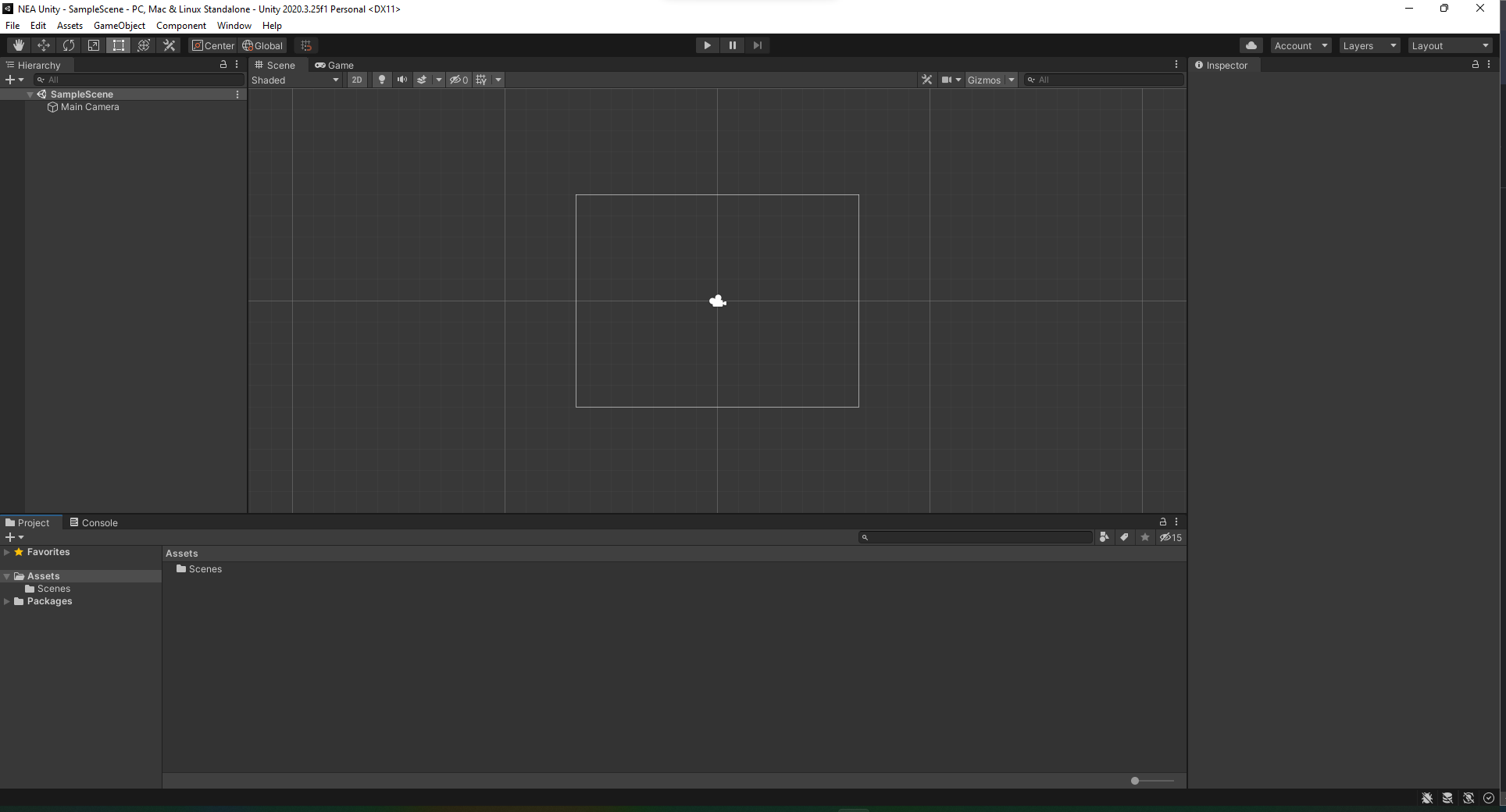
##### Language and engine

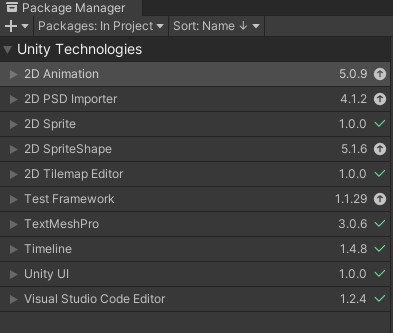
I am using the C# language and the Unity game engine. To edit code, I will be using Visual Studio Code.

Here are the coding conventions that I will be following:

<https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/coding-conventions>

##### CREATING THE PROJECT



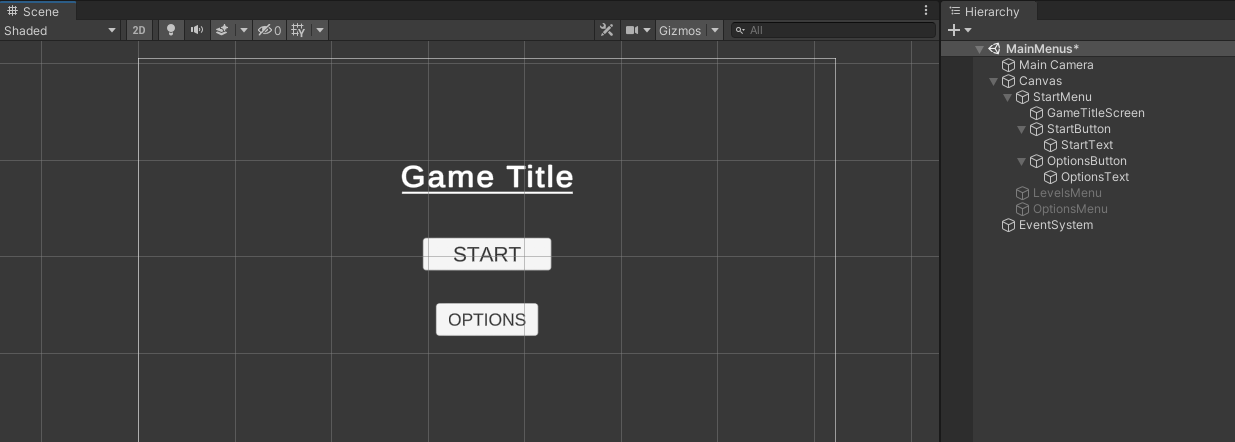


I created the Unity project using their 2D template, which comes with a few libraries to make working with 2D sprites and animations easier. I removed unnecessary libraries that will not be useful to making my game as they would clutter up my project folder.

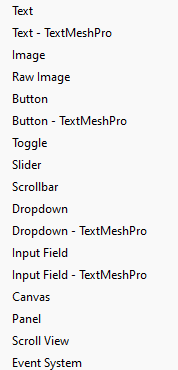
## MAIN MENU

#### Start menu setup

I started working on a basic start menus scene first. It will contain the main starting screen, an option screen, and a level selection screen. For this, I will rename the default SampleScene into a StartMenu scene.



Here is how my scene and my object hierarchy look like at the moment. The Canvas object holds all the User Interface objects, which can be of the following types:



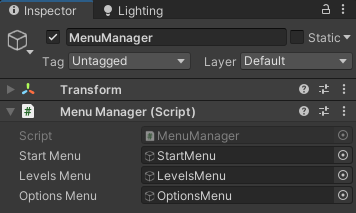
The StartMenu object holds the User Interface elements that appear when the game is launched (which are the buttons and text I just implemented).

Reference Video\_01.mp4 to see buttons being tested.

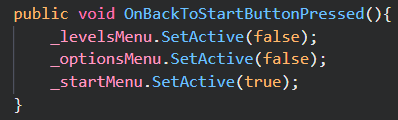
To implement behaviour to the buttons, I created a new script called MenuManager and attached it into a new object in the scene called MenuManager. This script will be responsible of handling the menu.



I set up three private variables that hold the reference to the three objects that each hold a different menu. The SerializeField attribute makes the variable show in the Unity inspector, so I can drag in the reference of the object I want into the appropriate field. The script derives from MonoBehaviour as it is a script that must be attached to a game object.



Then I wrote two functions that are called when the start and options button are pressed. Each of these will disable the start menu and enable the respective menu (the Start button will disable the start menu and enable the levels menu; the Options button will disable the start menu, and enable the options menu).

I have also implemented a button and a function to go back to the start menu. The button is the same both in the levels screen and the option screen and they call the same function.

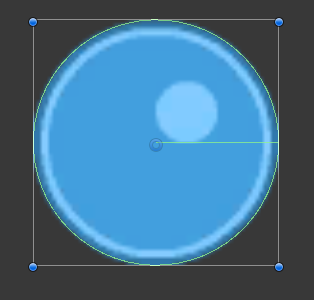
Refer to Video\_02.mp4.

## CREATING GAME MECHANICS

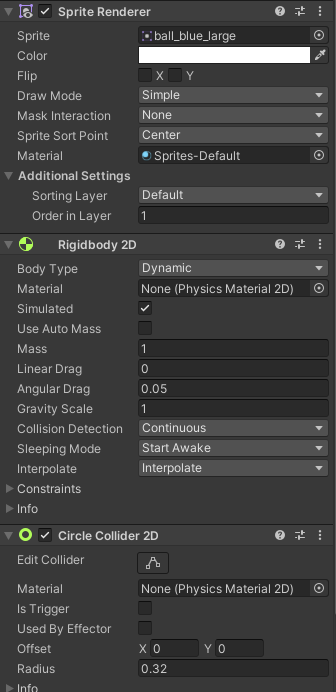
I created a new scene as a testing scene called TestLevel in order to create all the mechanics that will be present in the game. I then imported the game’s graphical assets. I downloaded them from this website:

<https://www.kenney.nl/assets/rolling-ball-assets>

###### Setting up the player



For the player’s ball, I created a new object and attached 3 Unity components to it.



The Sprite Renderer simply takes an image source and displays it on the screen on the position of the object. It also handles sorting layers when drawing the image on the screen. I set the order in layer to 1, as the image I am using as the background has an order in layer of 0, the player object will appear on top of the background.

<https://docs.unity3d.com/Manual/class-SpriteRenderer.html>

Rigidbody 2D is a component that handles simple 2D physics such as mass and gravity of an object.

<https://docs.unity3d.com/ScriptReference/Rigidbody2D.html>

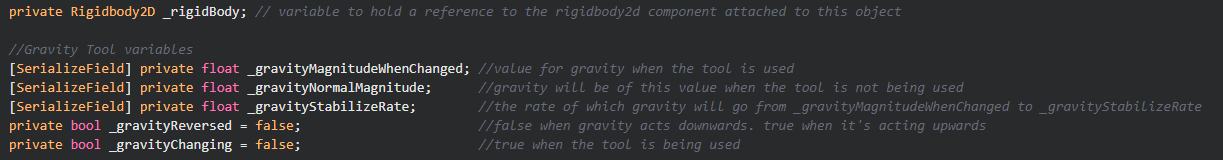
Circle Collider 2D is a component that adds a circular collider to an object and detects collisions.

<https://docs.unity3d.com/ScriptReference/CircleCollider2D.html>

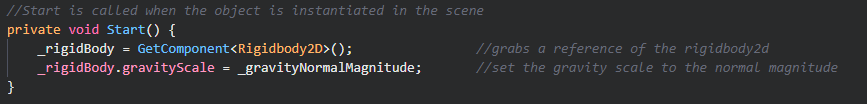
I then created a new script called Player and attached it to the player object. This script will handle the player input and behaviour. For now, I will write all the player code in this script, but later I will reformat it by creating multiple scripts that will each handle specific behaviour for the player, in order to keep my code organised and easily readable.

#### Gravity tool

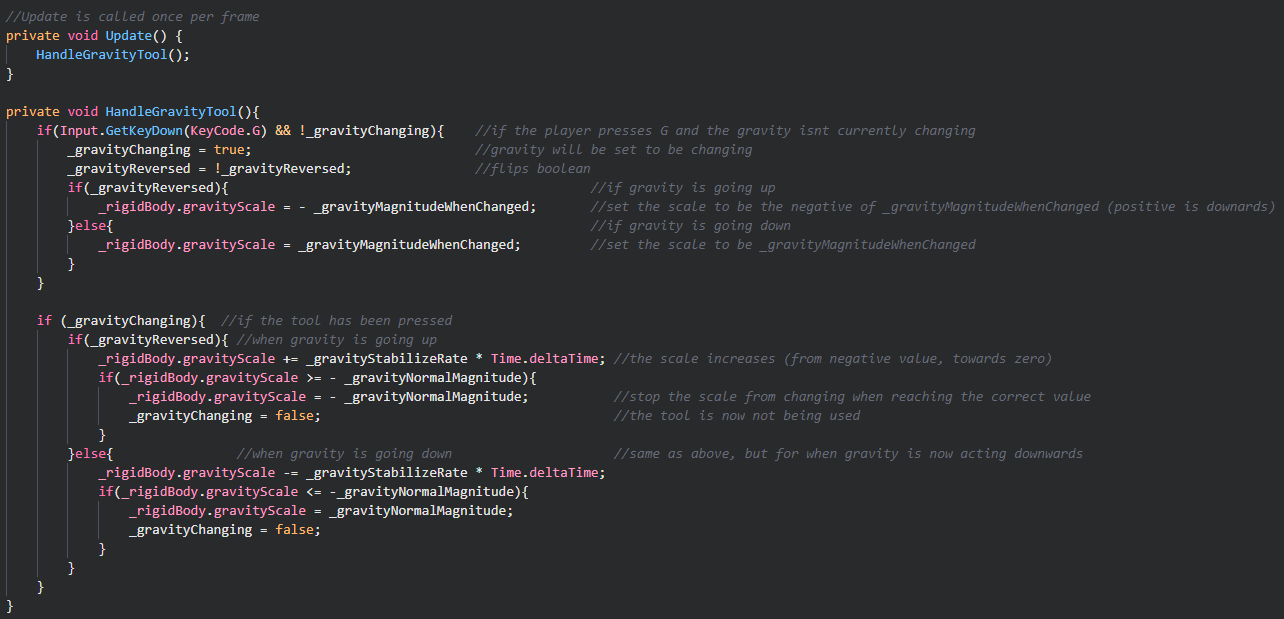
I started by creating the variables I needed. Those are the reference to the Rigidbody2D component, values for the gravity tool, and Booleans for the logic of the code.



Then I grabbed the reference of the Rigidbody2d component and set the initial gravity scale in the Start method (derived from MonoBehaviour).



I wrote all the logic of this tool in one function called HandleGravityTool and made sure to call it for each frame of the game.

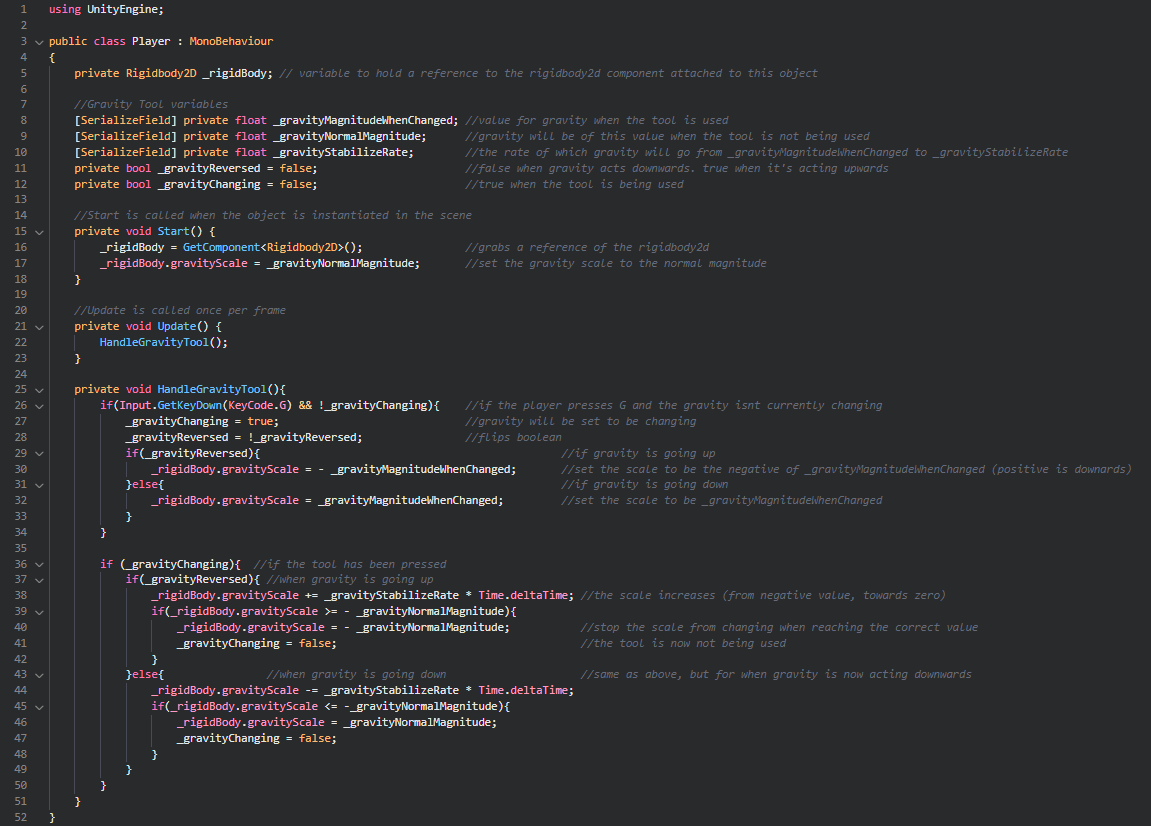


I added comments for each section of the code to make sure it is more readable.

Time.deltaTime is the time in seconds from the previous frame to the current one. I multiplied my stabilization rate to Time.deltaTime in order to keep the decrease/increase rate frame rate independent.

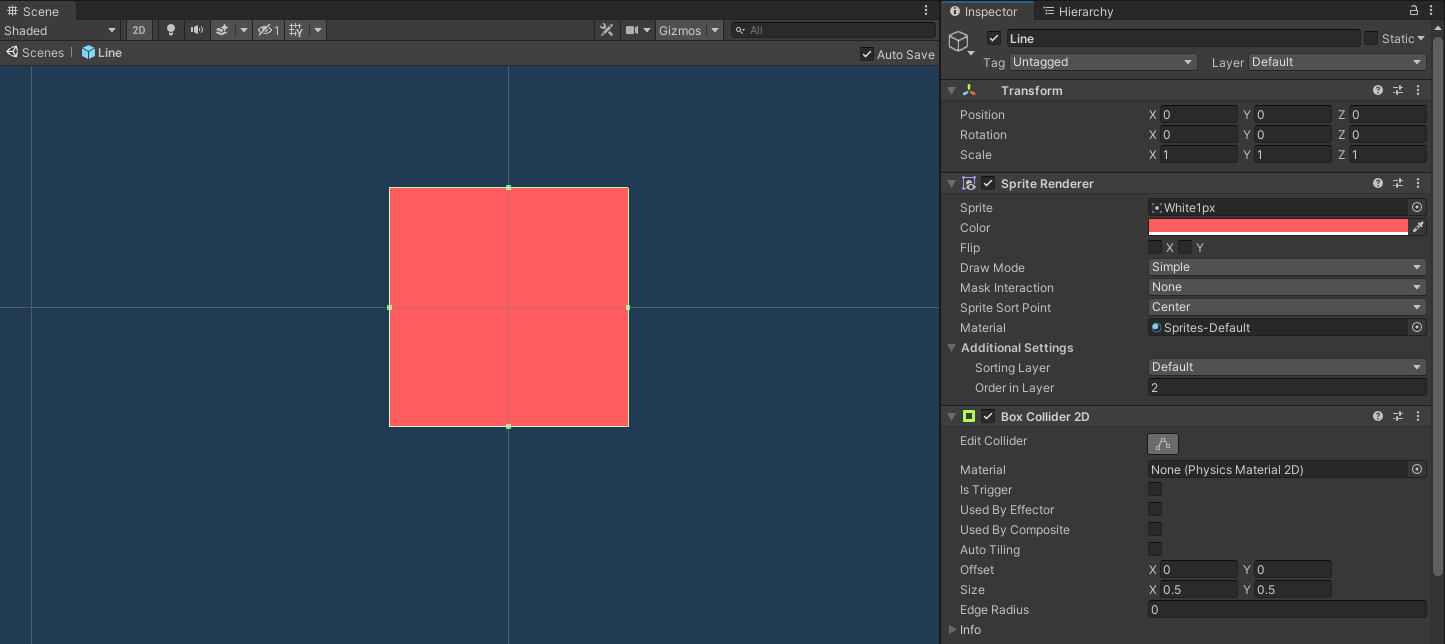
<https://docs.unity3d.com/ScriptReference/Time-deltaTime.html>

Here is the full Player script so far:

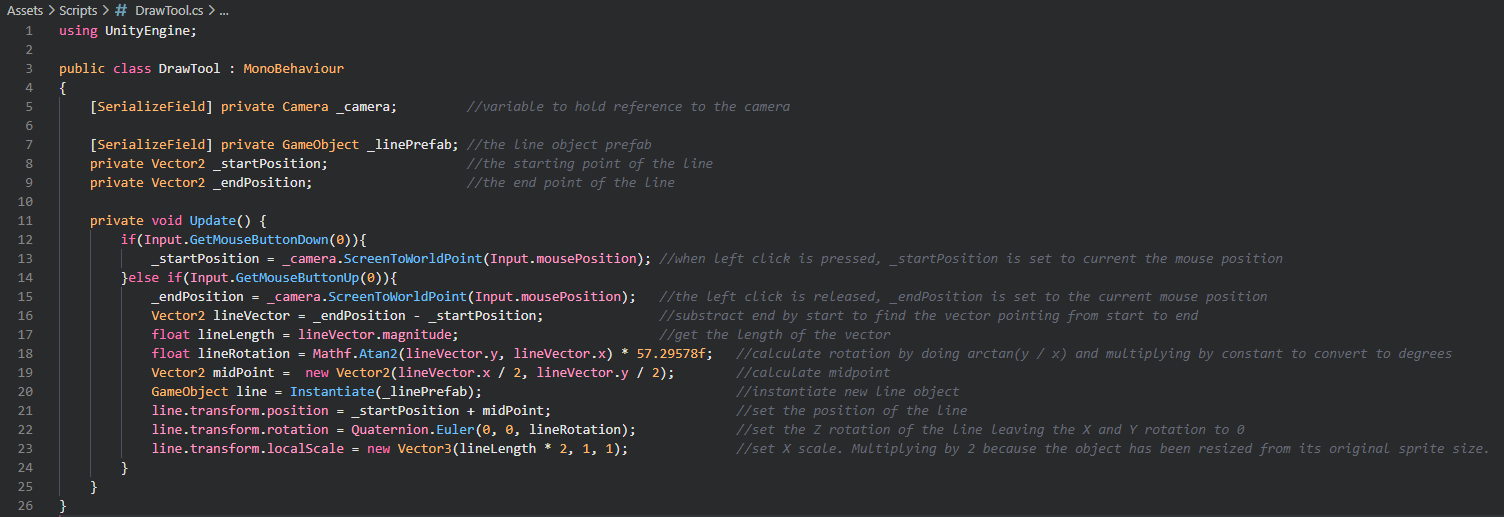
Refer to Video\_03.mp4 for testing. I am using a program called NohBoard ( <https://sourceforge.net/projects/nohboard/> ) to overlay my keyboard input and visualise it in the video. On the right side, in the inspector window, I can see the current gravity scale in the Rigidbody2D component. I used it to check if my code is changing it appropriately and it did.

#### Draw tool

I created a new script called DrawTool and attached it to the player object. First I needed to create a “blueprint” object for a line.



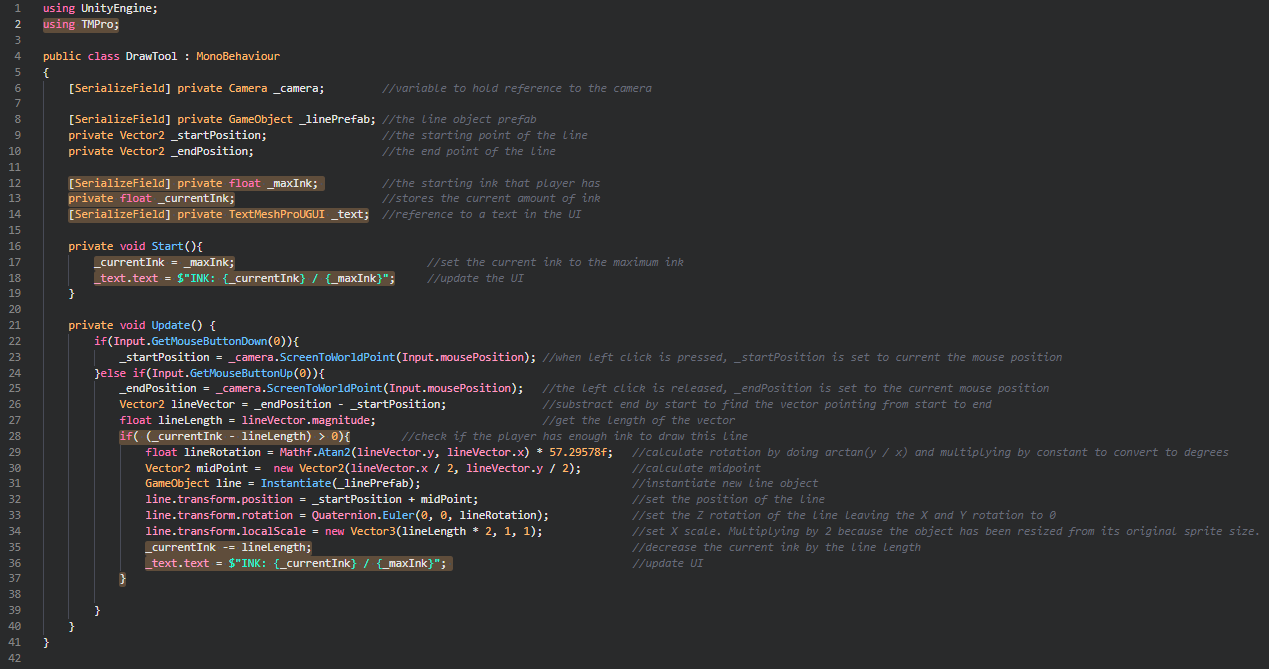
This is a prefab so it will be stored in the project files. It is a simple square sprite with a Box Collider 2D component that will allow it to have collisions. When the entire object scales in size, the box collider will also scale accordingly.



This is the full script. I first made variables to grab references to the camera and the line prefab. Then I made two global variables for the start and end position. In the Update loop, if left click is pressed down, the start position of the line is set to the current mouse position. The player then holds down the mouse and releases it where they want the end position to be. When releasing the mouse, the line is drawn on the screen.

Refer to Video\_04.mp4. I am using the same program as before to show mouse input.

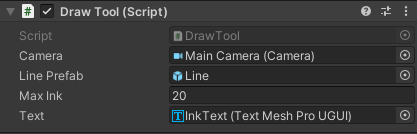
Since I do not want the player to be able to draw as many lines on the screen as they want, I need to limit the number of lines they can draw.



The highlighted lines are the ones I added to implement this feature.

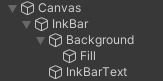


I added a UI text element to display the ink remaining. It says “Ink Text” when the game is not running.



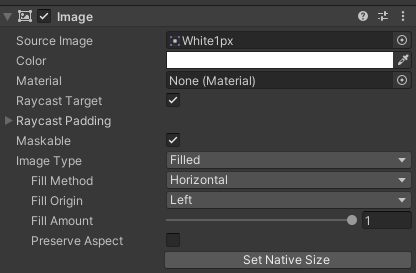
These are the Serialized fields that I set in the inspector for the Draw Tool script.

Refer to Video\_05.mp4. As shown in the video, the ink is reduced according to the length of the drawn line. When the ink got low, I was not able to draw a long line. The UI correctly updates; however, I do not find it intuitive to read a long decimal number, so I decided to replace the text element with a bar.

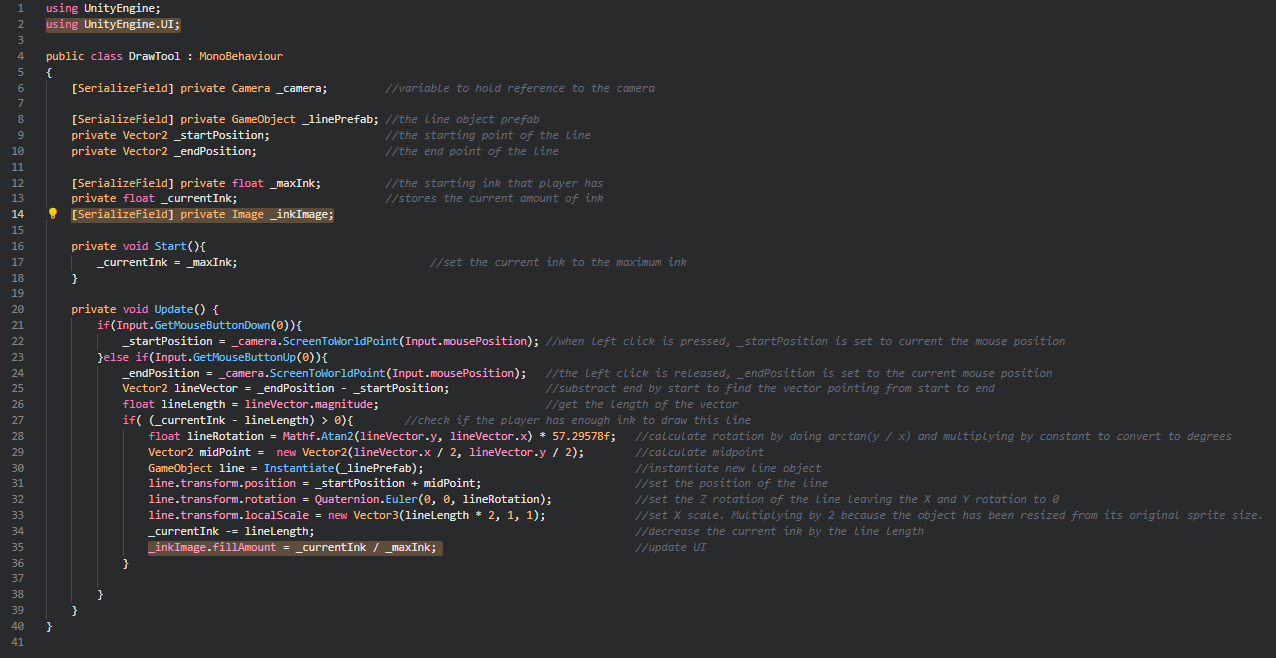


I deleted the InkText object and added these instead. The InkBar object holds the UI elements regarding the UI for the Draw Tool. The Background object is a simple black image for the background of the bar. The fill object is a white sprite and will be the one edited at runtime when the player uses ink. InkBarText is a simple text object so the player knows that the slider represents ink.

Here is how it looks.



This is the image component of the Fill object. The most important fields here are the Image type, and Fill amount. I set image type to be Filled horizontally and from the left. The fill amount variable ranges from 0 to 1, so I will need to calculate the percentage of ink remaining, and apply it to this variable.

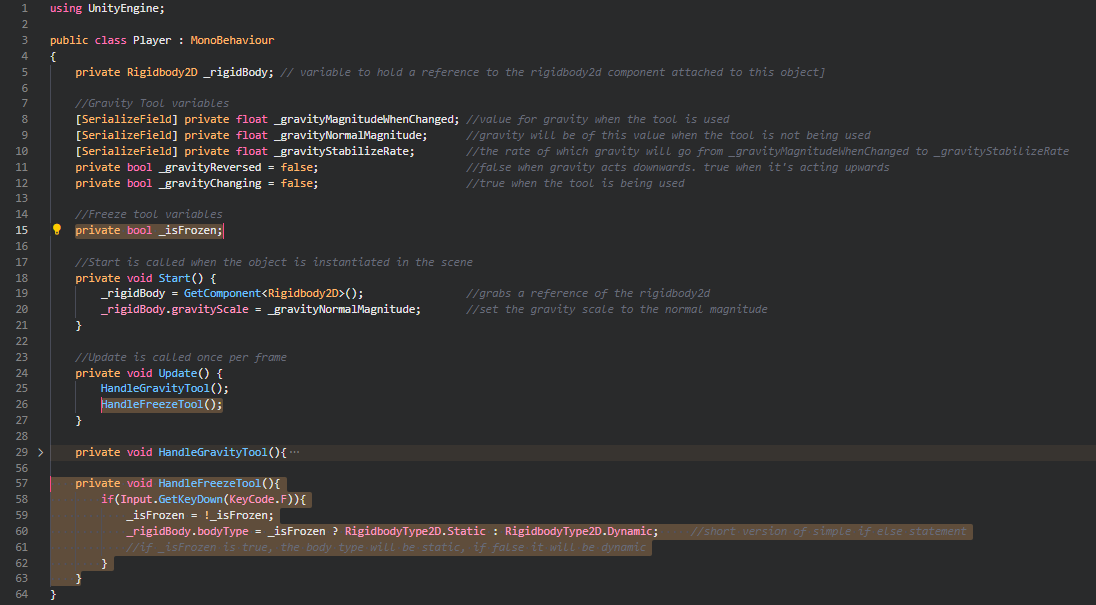


I highlighted the lines I changed. I removed the code for the UI text as I will not be using it anymore.

Refer to Video\_06.mp4. As shown in the video, the Ink bar works as intended. I am happy with the result. It seems more intuitive than the UI text I implemented earlier.

#### Freeze tool

I will implement this in a similar way I implemented the gravity tool: with a press of a key, the player can freeze the ball’s movement and freeze it mid-air. I will write this tool’s code in the Player script.



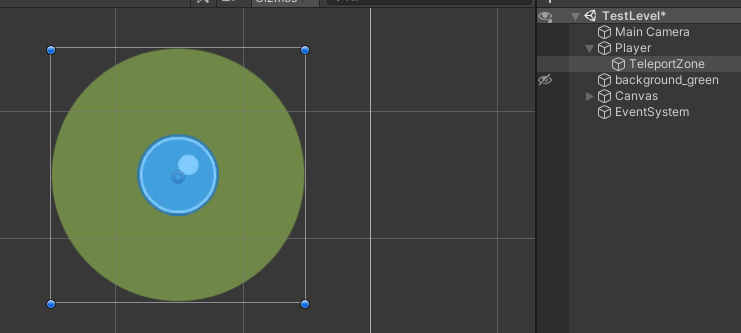
Here is the code I wrote for this tool. I added a Boolean to keep track if whether the player is frozen or not. I created the HandleFreezeTool function and made sure to call it in the Update function. When pressing the F key, the Boolean is flipped. If \_isFrozen is true, the Rigidbody’s body type is set to static, otherwise (when \_isFrozen is false) it is set to Dynamic. I used a ternary conditional operator as it is simply a short version of an if else statement with two outcomes.

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/conditional-operator>

Refer to Video\_07.mp4. As shown in the video, the tool works as intended and when the ball is frozen, the gravity tool can still be used so these two tools do not interfere with each other.

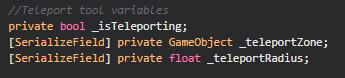
#### Teleport tool

I will write this tool in the Player script, in its own function just like the gravity and freeze tools.

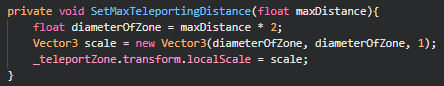


First, I added a sprite called TeleportZone as a child of the Player object. This way its position will follow the player’s position. I added a circular sprite and lowered the alpha value in order to make it semi-transparent. The purpose of this sprite is to show where the player can teleport to. I will change the size of this sprite depending on the maximum distance the player can teleport. On more difficult levels, I will reduce that value therefore shrinking the TeleportZone's sprite.

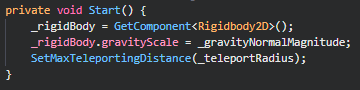
I created the variables needed for this tool:



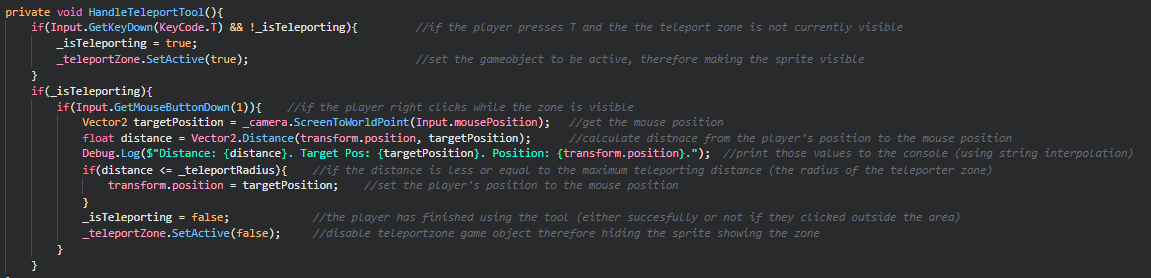
\_teleportZone is a reference to the TeleportZone game object in the scene hierarchy. \_teleportRadius is the maximum distance the player can teleport to. I wrote a function to change the size of the teleport zone based on the maximum distance the player can teleport to.



It takes a parameter of type float and it has no return. When setting the scale of the circle, the diameter is needed, so the max distance is multiplied by two and the result is stored in a local variable of this function called diameterOfZone. A variable of type Vector3 (Vector 3 contains three floats: x, y and z) uses this diameter to calculate the appropriate scale for the zone. The reference of the zone object is then accessed to change its scale to the just calculated new scale.

I call this function in Start to set the scale of the zone. I will call this function when different levels with different difficulties will require the teleport radius to be changed.

Next, I implemented the logic of the tool:

When pressing T, the zone will be activated and shown and if the player right clicks inside that zone, they will be teleported inside it. If they click outside the zone, nothing will happen. After clicking, whether inside or outside the zone, the zone will return to be hidden.

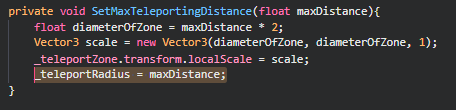
The “Debug.Log()” line prints the Distance, target position and the current position to the unity console window. I used this to test if the code was working correctly. I used string interpolation in order to clearly see which value was displayed.



I set the teleporting variables in the inspector.

Refer to Video\_08.mp4. As shown in the video, I used my freeze tool to freeze the ball for easier testing. The values are displaying in the console window. When the distance value is higher than the teleport radius, the player is not teleporting.

One thing I missed when creating the function to set the scale of the zone, is to apply the parameter to the teleport radius variable.



I added this line to the function.

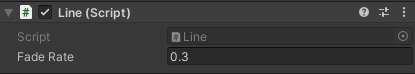
#### Fade lines over time

After letting my peers and family test the mechanics so far, I found that the most used tool is the draw tool, which gets used until the entire screen is cluttered with lines. To avoid players breaking a level, or trapping themselves, I decided to make the lines slowly fade over time and then disappear.

For this, I will create a new script and attach it to the Line prefab.



This is the Line script I created for the Line prefab. Each line that will be instantiated will contain this script.

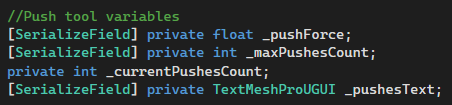


I tried a few different values for the fade rate and 0.3 was the one that made the line last just long enough. In the future I may change it based on feedback.

Refer to Video\_09.mp4. As shown in the video, the line fades correctly, and when it is no longer visible, it is destroyed therefore allowing the player to pass through normally. This addition does not interfere with the draw tool.

#### Push left and right

Currently the player’s only way to move themselves left and right is by creating slopes with the draw tool. I am adding a new tool in order for them to be able to push the ball left and right, at a limited amount. I will write this tool in the player script.

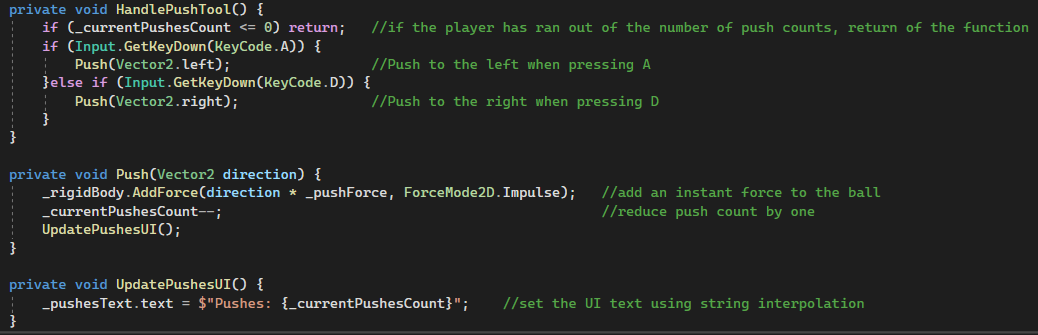


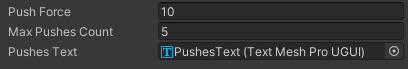
These are the variables I added for this tool. The push force is the magnitude of the force applied on the ball.

\_pushesText is a reference to a Text object in the User Interface Canvas.



I added a using statement at the top of the script in order to be able to set the UI text.

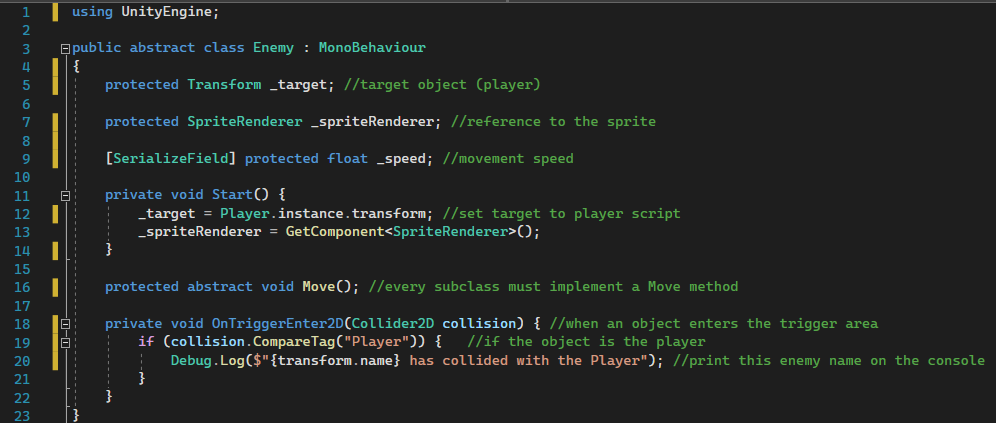
Here are the three functions for the push tool.



And these are the variables I assigned in the inspector.

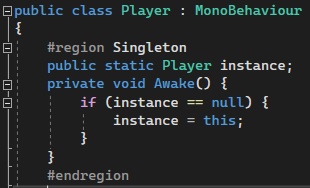
#### Enemies

For the enemy entities in the game, I made a base Enemy class that each enemy will derive from.



Most variables in this script have a “protected” access modifier. This is similar to private, except that all subclasses of this master class can read and write the variable.

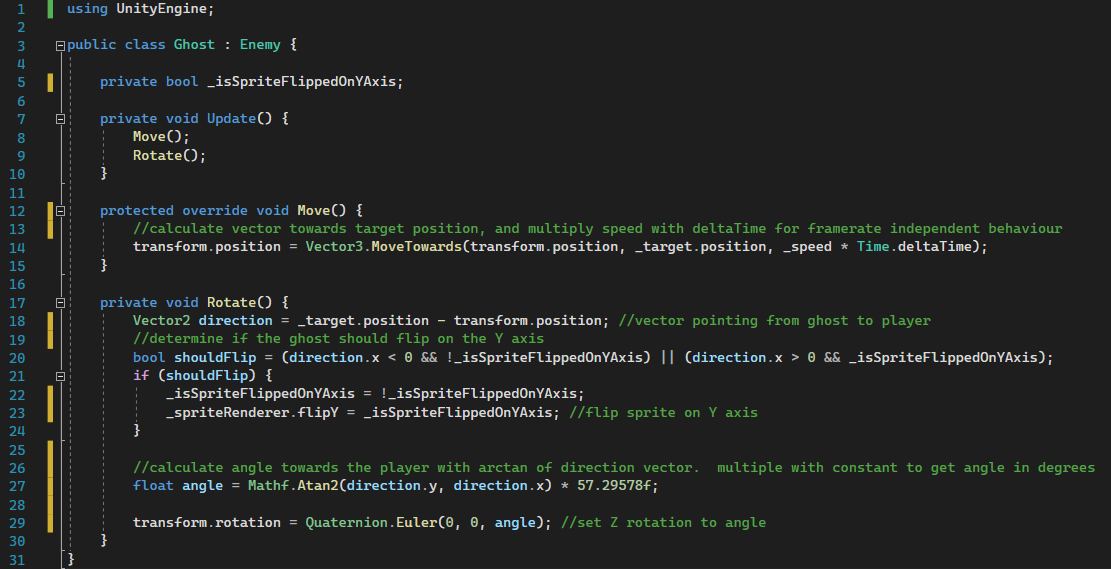
To get the reference to the player, I added the Singleton pattern to my Player script.



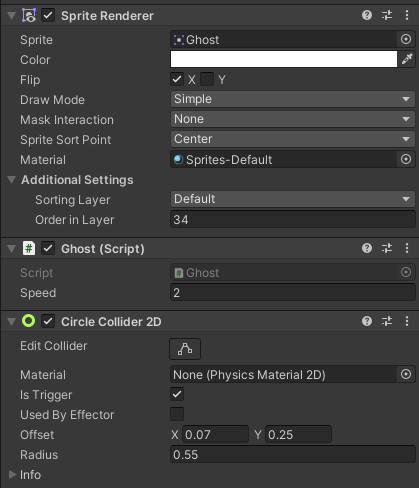
This is how it looks like at the top of the script. I create a static instance of the class inside the class itself, and assign it to the class. This also makes sure that there is always only one instance of the player class. Because it is static, any script can access the Player class now without having a reference variable first.

* **Ghost**

The first enemy I made is a ghost enemy. It will chase the player, ignoring all collisions.



It derives from the Enemy class. The Move method implemented has the override keyword in order to override the empty behaviour from the abstract Move method in the super class. The Rotate method rotates the ghost accordingly (towards the player).

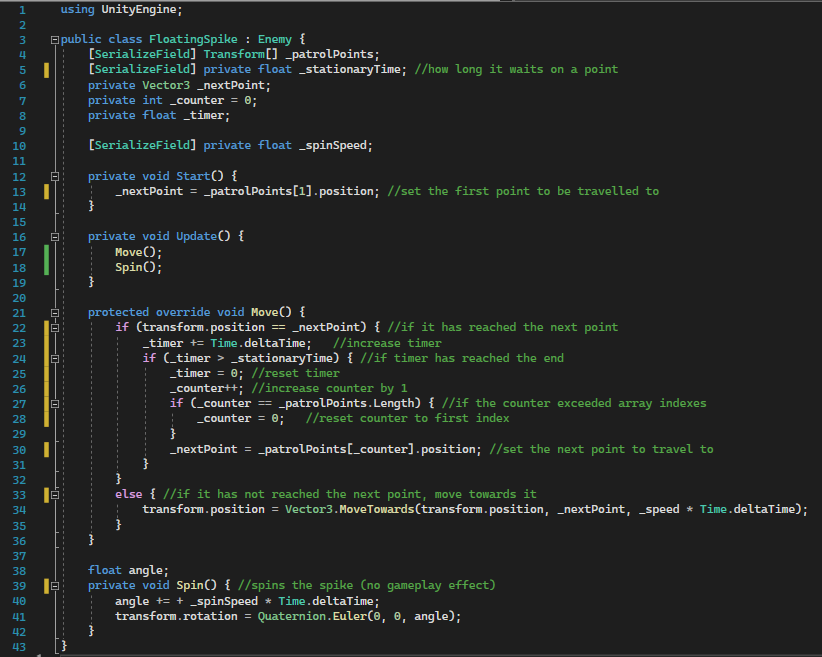


This is the Ghost object and its components.

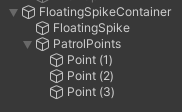
Refer to Video\_10.mp4. The ghost follows and rotates accordingly.

* **Floating Spike**

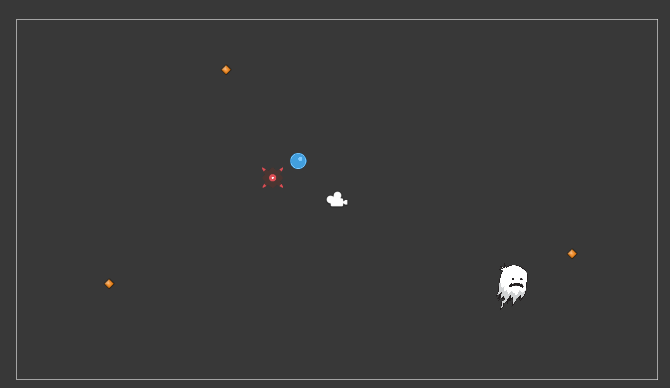
I made an enemy that patrols predetermined points in the level. It stays on a patrol point for a certain time and then moves onto the next one. After reaching the last point, it will go back to the first point.



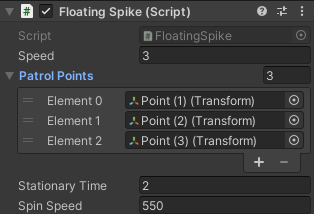
In the scene hierarchy, this is how this enemy is set up:



The FloatingSpike object is the actual enemy object. PatrolPoints contains all the patrol points, which are just empty objects with a position. These two objects are contained in FloatingSpikeContainer, for better organisation and easier porting to other levels.



The red object is the spike itself. The orange icons are the patrol points. The icons are not visible in games as they are a feature in Unity that allows to visualise empty game objects with an icon in the scene view.

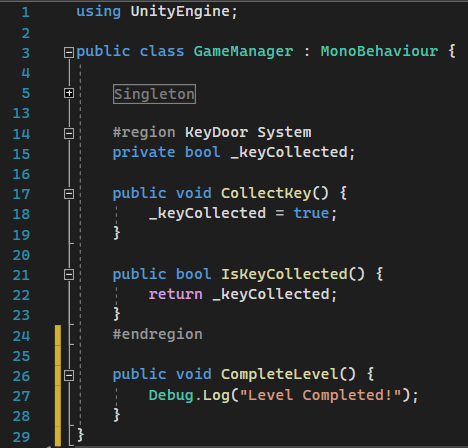
These are the values set in the inspector.

Refer to Video\_11.mp4. As shown in the video, the spike travels to each patrol point in order (and loops after the last one) and also waits 2 seconds at each point before moving to the next one, and also spins on itself.

#### Key and door system

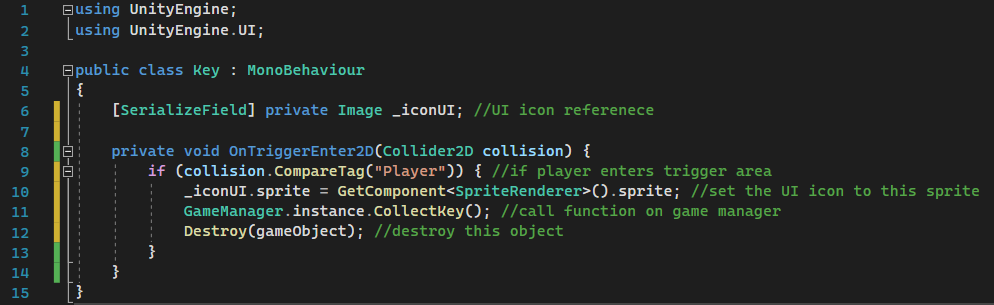
For certain levels to be completed, the player will need to gather a key and use it to open a door.

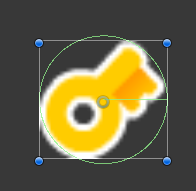
First, I created a GameManager script to handle the level behaviour.



This script is attached to an empty game object in the scene hierarchy.

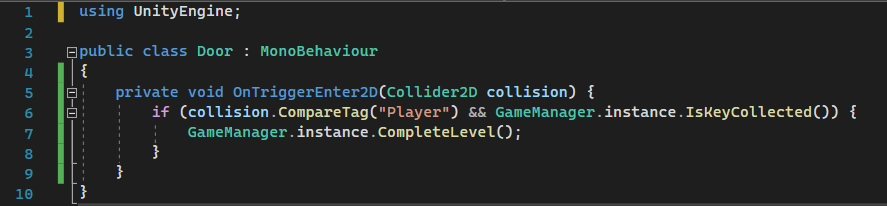
Then I implemented a Key script and attached it to a Key object in the scene.





The key icon in the user interface (image on the right) is an outline of the key sprite. When the key is collected, it will be set to the yellow key sprite.

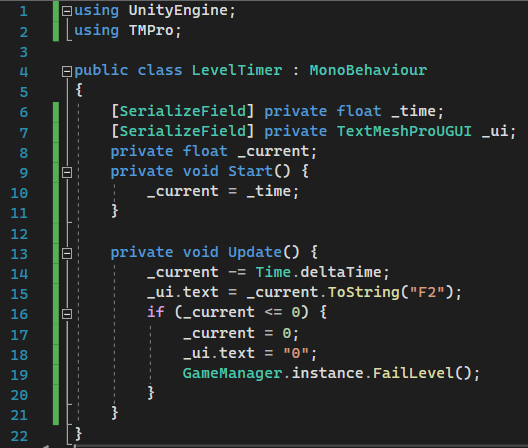
I then created a Door script.



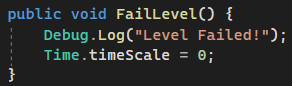
Refer to Video\_12.mp4. As shown in the video, when the key is collected the UI icon updates. When the player collides with the door, because the key is collected, the message of level completion is printed to console (appears on the bottom left of the video as well).

#### Level timer

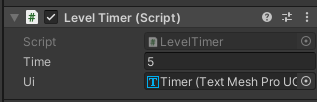
For some levels, instead of a key and door system, the player will have to survive for a certain time.



This script is attached to the GameManager object. In levels in which I need a level timer, I will add this component to the game manager script. ToString(“F2”) sets the timer text to the value to two decimal places.



I also added a function to fail the level in the game manager script. It will freeze the game when called.

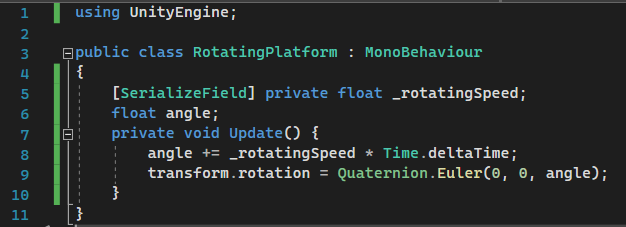


These are the values set in the inspector for the level timer. The UI field is a reference to a text object in the UI.

Refer to Video\_13.mp4. As shown in the video, the timer updates correctly and when it reaches 0, the game freezes.

#### Rotating platforms

These are simple platforms that keep rotating over time.

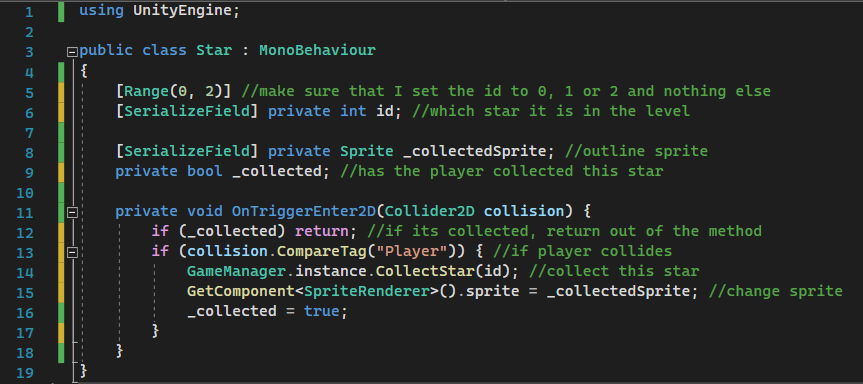


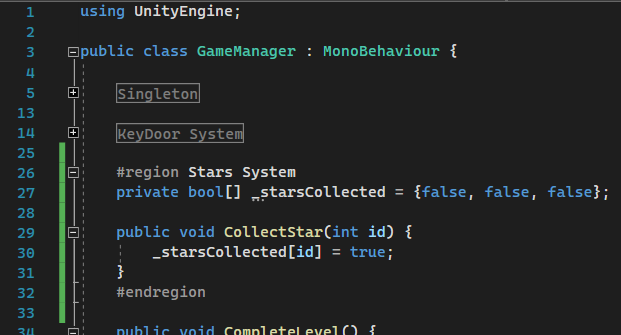
The code for this is exactly the same as for the spinning of the floating spike.

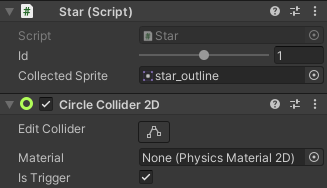
Refer to Video\_14.mp4.

#### Stars system

To incentivise players to try the levels again, I am adding a classic ‘stars’ system. In each level, there will be three stars that can be collected and they can be used in the shop to change the appearance of the player.





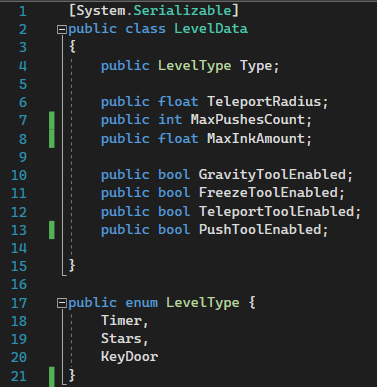


The \_starsCollected array’s data will be used to display to the player which stars they collected and also it will be used to add the number of stars collected in that level into the number of stars collected throughout the game.

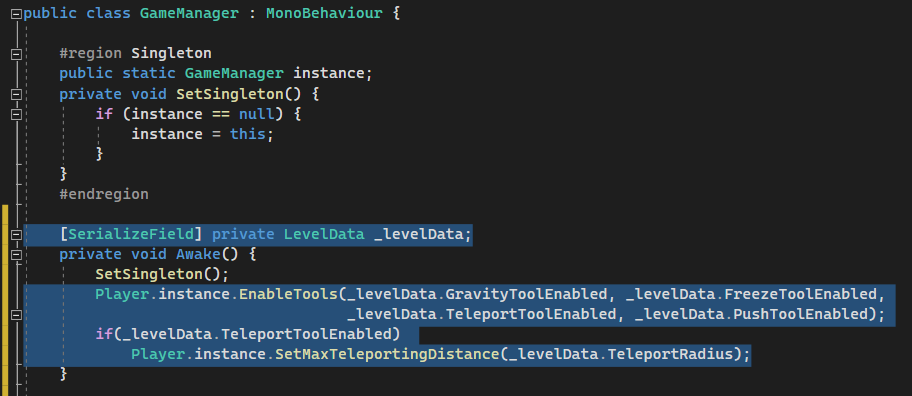
Refer to Video\_15.mp4.

#### Level data

So far, the game has been developed on a single test scene. Each level will be its unique scene, with each having its own properties such as whether some tools are enabled or not.

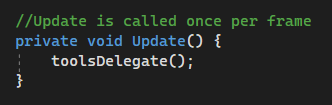


I made a new script called LevelData and set it to be serializable in order to change every variable in the inspector. This script also contains an enumerator called LevelType.



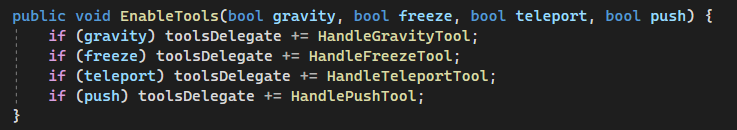
I added this code to the Game manager script. It holds a LevelData class that will show up in the inspector and will be able to be edited. It calls a function on the player script that enables the tools accordingly (they are all disabled by default). To enable and disable tools, I used C# delegates.

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/>



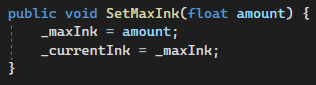
I added the delegate at the top of my player script, and in the Update function, instead of calling every tool handling function, I call the delegate.

The EnableTools function shown above is this:

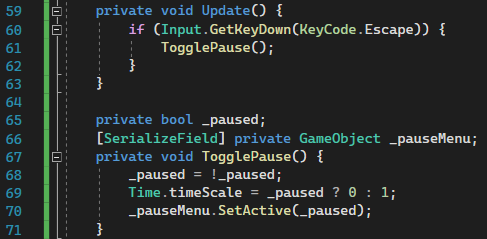


It takes a Boolean for each tool (except ink as it will always be enabled). If the bool is true, the delegate subscribes to the corresponding function.

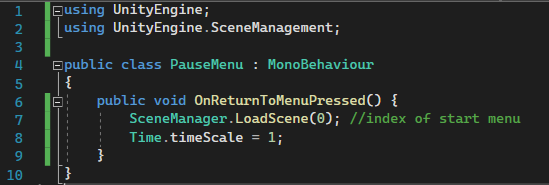
I also added a function to change the maximum ink for the ink tool.



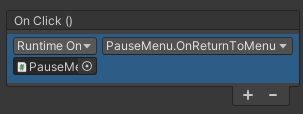
#### Pause system



I implemented the pause system in the Game Manager script. The pause menu object contains the UI elements of the pause menu.

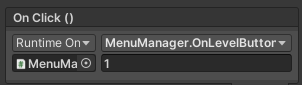


I attached this script to the pause menu object. The function will be called when the button to return to the start menu is pressed.



Refer to Video\_16.mp4. As shown in the video, I built a new level using the objects from the test level. This will be the first level and in the Unity build settings, it has an index of 1 (0 is the main menu as shown in the pause menu script). I also made some buttons in the level selection screen and when pressed, they call a function on the menu manager script:



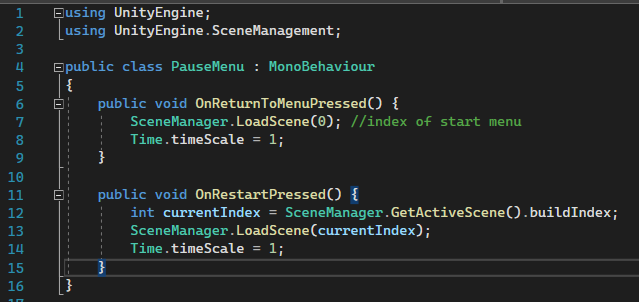


The integer that is passed is set in the inspector for each button.

The pause menu works as intended. It freezes the level when pressing the escape key and resumes it when pressing it again.

#### Building the first level

I added more UI buttons and functions to the existing scripts.



I added a ’Level Completed’ screen and ’Level Failed’ screen. The first one has a button to proceed to the next level. The second has a button to restart the level. I added a ’Restart Level’ button to the pause menu as well.

When completing a level, the UI displays which stars have been collected.

Refer to Video\_17.mp4 for victory screen.

Refer to Video\_18.mp4 for failed screen and restart buttons.

The tiles in the level were created using Unity’s tile map system (no coding required).

#### Persistent data saving

The player will be able to use the stars collected to purchase additional balls. At the moment the game does not feature persistent saving, therefore when the player closes the game, they lose all their progress. I will also use persistent data to keep track of the completed and locked levels.

# 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