NEA Project

Stealth Game

*By Hamzah Mohamed*

Table of Contents

[**Analysis**](#_x01mm1eymois) **3**

[Project introduction - summarised](#_8u4qgpj8rake) 3

[Project introduction](#_49l9svj59dk3) 3

[Questionnaire](#_4n0lcnn8cva3) 4

[Objectives](#_3lnu1lxikhcv) 8

[Pillars](#_egysiixx2jbl) 10

[Prototypes](#_6txhxwodwj9p) 10

[**Documented Design**](#_qlie79hk628z) **11**

[Overview](#_fx7rvsrapuil) 11

[Notes/Drafts](#_79pzolss265j) 12

[Flowcharts](#_13eeo0apqigo) 16

[Player Sprite](#_gu90ob9krrjh) 20

[Data Structures](#_b8ud8emgeefe) 21

[Pseudocode, Algorithms, & Mathematical operations](#_3s6v825qcqdi) 22

[UML Class Diagrams - Individual](#_kjx1mv1endfe) 30

[UML Class Diagrams - Relationships](#_rituyzgsrjgj) 51

[**Technical Solution**](#_ch0s048xfvli) **53**

[Player.cs - R](#_izaa2j2hj5as) 53

[PlayerMovement.cs](#_7sjf3r6ba3gj) 55

[PlayerDamage.cs](#_si9tp06kzmjt) 56

[Weapon.cs](#_pd4nlgr8nlng) 57

[PlayerNoise.cs](#_ii80l6ejrjus) 59

[Enemy.cs](#_fbbxqcqwznk1) 61

[EnemyDamage.cs](#_5714pyx66ydx) 62

[Health.cs](#_148sesxilrk7) 63

[CameraMotion.cs](#_s4s50ncgsex1) 64

[Area.cs](#_4kp3eiuwr81h) 66

[MapManager.cs](#_n4paf2xds5w0) 67

[HighScoresManager.cs](#_pt4emgn99c87) 69

[isPause.cs](#_wu7o1j2f7228) 71

[GameManager.cs](#_3xtrlm8gua3i) 72

[NPC.cs](#_1x40j8d1q4wn) 74

[ChestTutorial.cs](#_qp31cpylop8k) 75

[Chest.cs - Some R](#_uco1q51k3axt) 75

[Portal.cs - Some R](#_1z56m7kkqb11) 76

[Colliding.cs - R](#_y80i8x4eo94c) 77

[Collecting.cs - R](#_xgtcwlvm73gs) 78

[TextOnScreen.cs - R](#_9bs0vmdqstoq) 79

[TextOnScreenManager.cs - R](#_z8jx8ae86sca) 79

[**Testing**](#_bisiuw5x0ts3) **81**

[Sound and player movement system](#_tod4bz864o5p) 81

[Collision mechanism](#_gdjttp9pc3i2) 82

[Text](#_swtquo7u5wpd) 82

[NPC collision](#_5lk79q2tfyhl) 83

[Sprite animation - chest](#_j4aa285lnook) 84

[Objective-specific](#_sxfnu4e04s1z) 85

[Final Test](#_hjv78m6h4l74) 88

[**Evaluation**](#_z11n3ua0kmi3) **89**

[Objectives](#_45by07s1fme1) 89

[Solution as a whole](#_ber4ipgz36x) 92

[**References**](#_xjjdcmmgt9r3) **93**

# Analysis

## Project introduction - summarised

I am attempting to make a small 2d pixelated game to interest my friends and family into playing games of similar genres to it. The main theme throughout will be stealth like combat. I want my game to give my friends and family a chance to experience a game where the storyline is linear and there is only one ending. The type of games I want to interest them in are those that use lots of planning and tactical play as opposed to the rogue-like and dungeon crawler types (where you go gung-ho when there are enemies around) that are currently popular among them and the rest of the playerbase.

## Project introduction

According to *Steam* (arguably the most popular video game digital distribution service) and *metacritic* (a website where people rate/review games they’ve played) , some of the most popular pixel graphic games currently are Stardew Valley, Terraria, Dead Cells, Noita and Enter The Gungeon. The most notable genres within these games are roguelike, dungeon crawler and rpg. These genres are often seen within most of the pixel graphic games released currently due to their growing popularity which leaves other genres of games untouched by the majority.

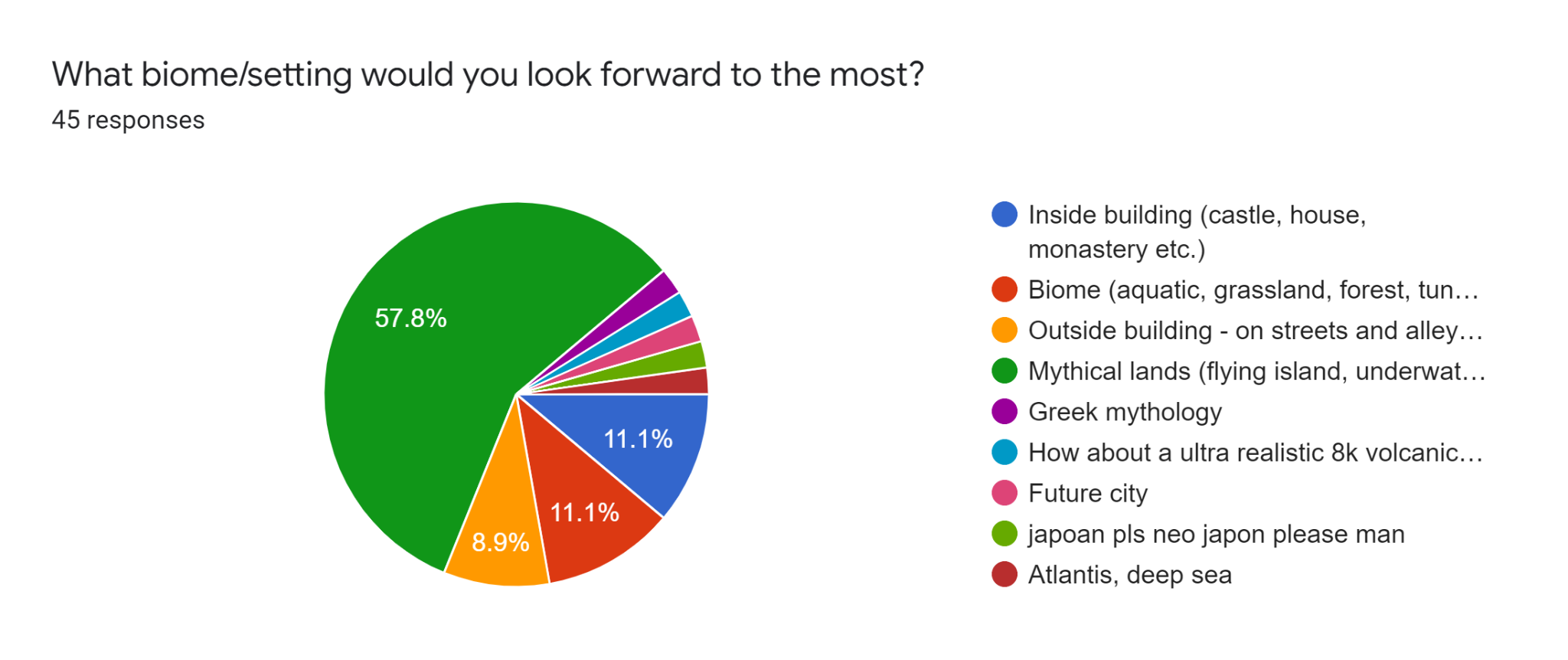
This is the problem that I’m trying to solve - to make pixel games with more niche and rare genres both appealing and intriguing to the current player base - more specifically my end user. The genres I have chosen to use are stealth and tactical play, genres that in my opinion and experience among my colleagues and friends do not receive enough recognition. Naturally, this means that my classmates/friends will be my target audience for this game as this game will attempt to introduce them to the world of pixel graphic games (to those who have not played these games before) and the rarely seen stealth/tactical play genres within these games.

My end users being my classmates presents me with advantages and disadvantages, of which the latter I will talk of here. One of the challenges I will be facing due to the nature of my end user - that they are mostly teens and young adults who enjoy games that are more fast paced and require less thinking and more ‘doing’ - is that I must be able to attract them to play a game that varies greatly compared to what they are used to or like, and make them grow an interest and develop a fondness for games of this genre (and other genres that are less relatively seen than the popular ones). To solve this problem in its essence would not be possible as this constitutes the whole premise of my problem, however, I can make the game seem more to their liking and more attractive to them by basing other aspects of this game on what they themselves want.

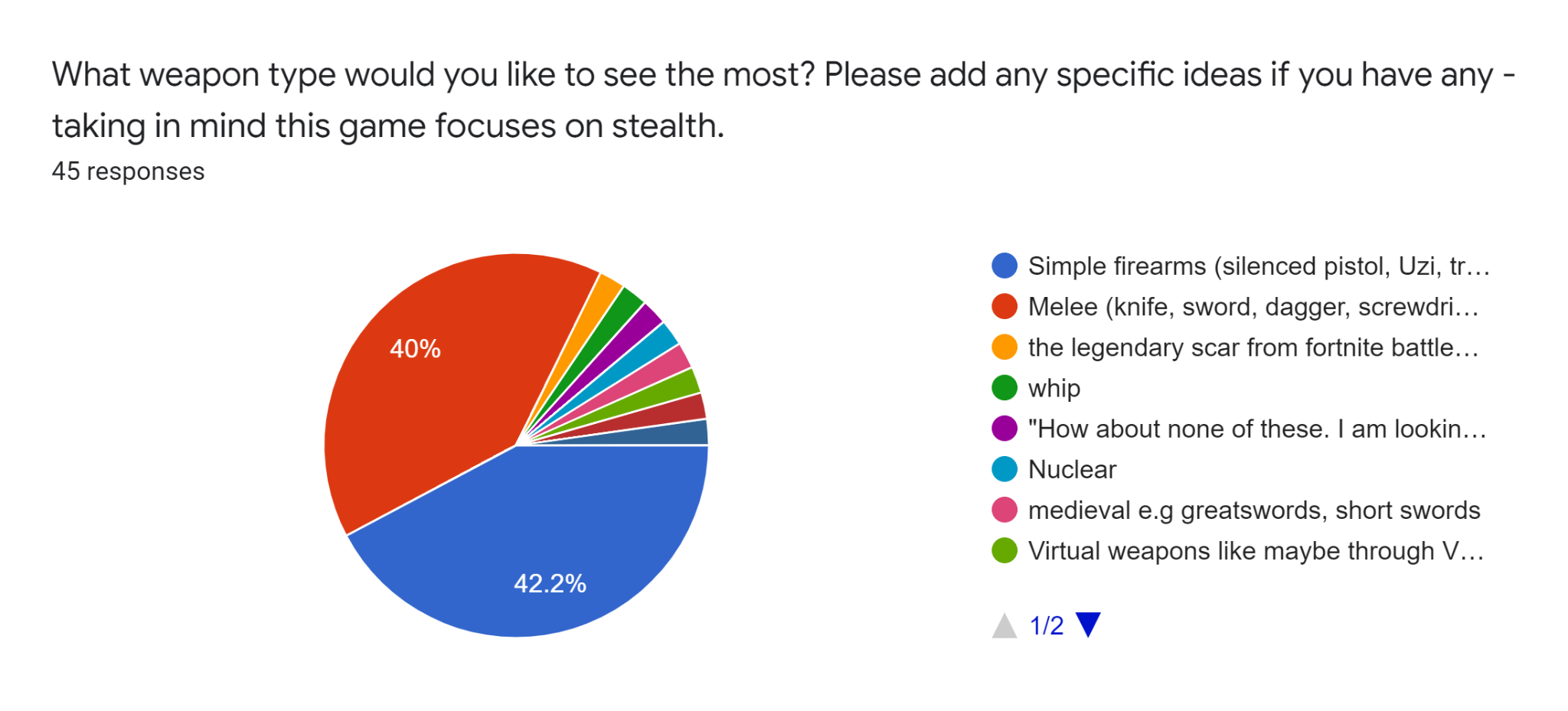
Another challenge I will be facing due to my end user is the less competitive environment that they are used to. Although no statistics are here to represent this, I know that many of my friends/classmates (and other young adults) in general are used to and prefer the competitive environment of multiplayer games, which unfortunately, will not be a feature available in this game. To compensate for this, I will introduce a stopwatch which will calculate how long the user took to complete the game or a highscore system that awards a highscore based on completion of the game. This will hopefully promote my classmates/friends to think outside of the box when playing and attempt to achieve the lowest time/highest score among themselves and their classmates - allowing my users to still experience the competitive environment of multiplayer games while not playing a multiplayer game.

## Questionnaire

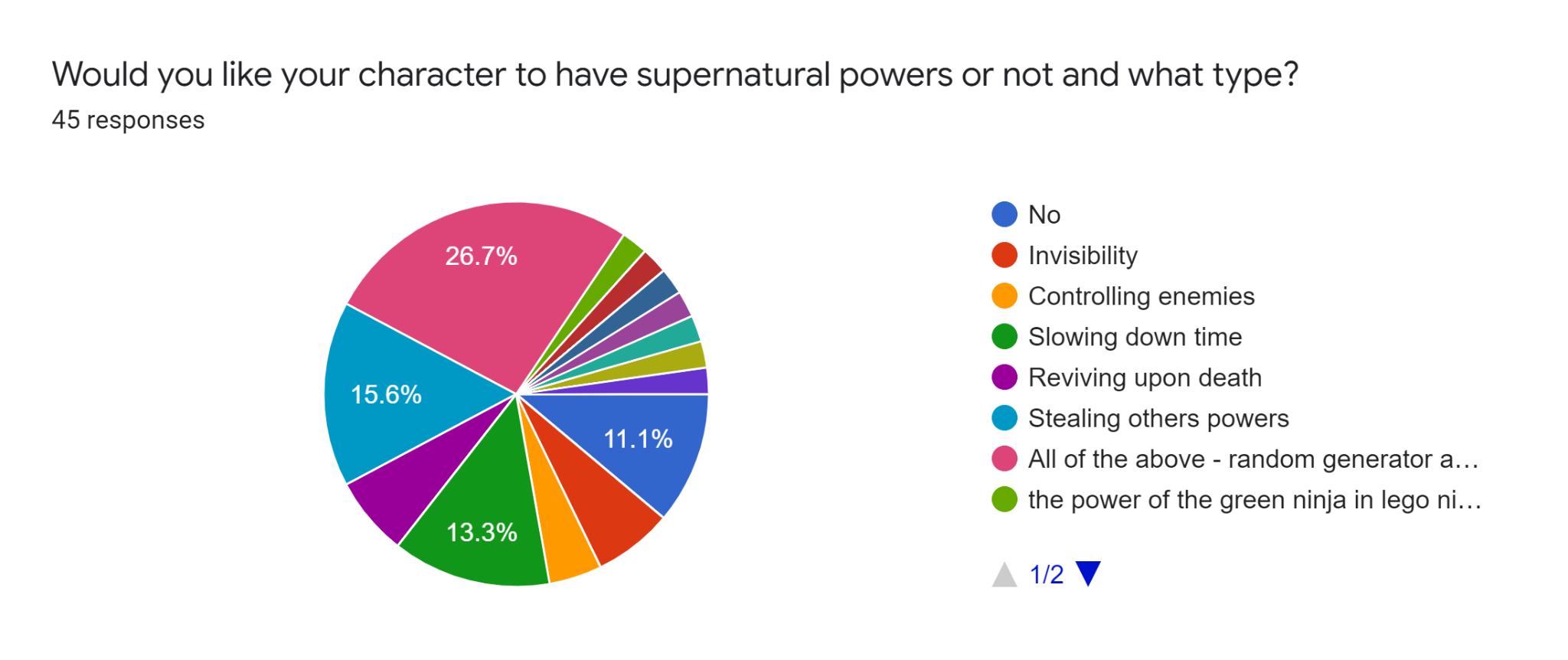
I made a questionnaire to find out more about my end-users requirements and preferences overall so that I can make them feel that the game was more dedicated to their specific wants - as opposed to using the general player bases preferences.



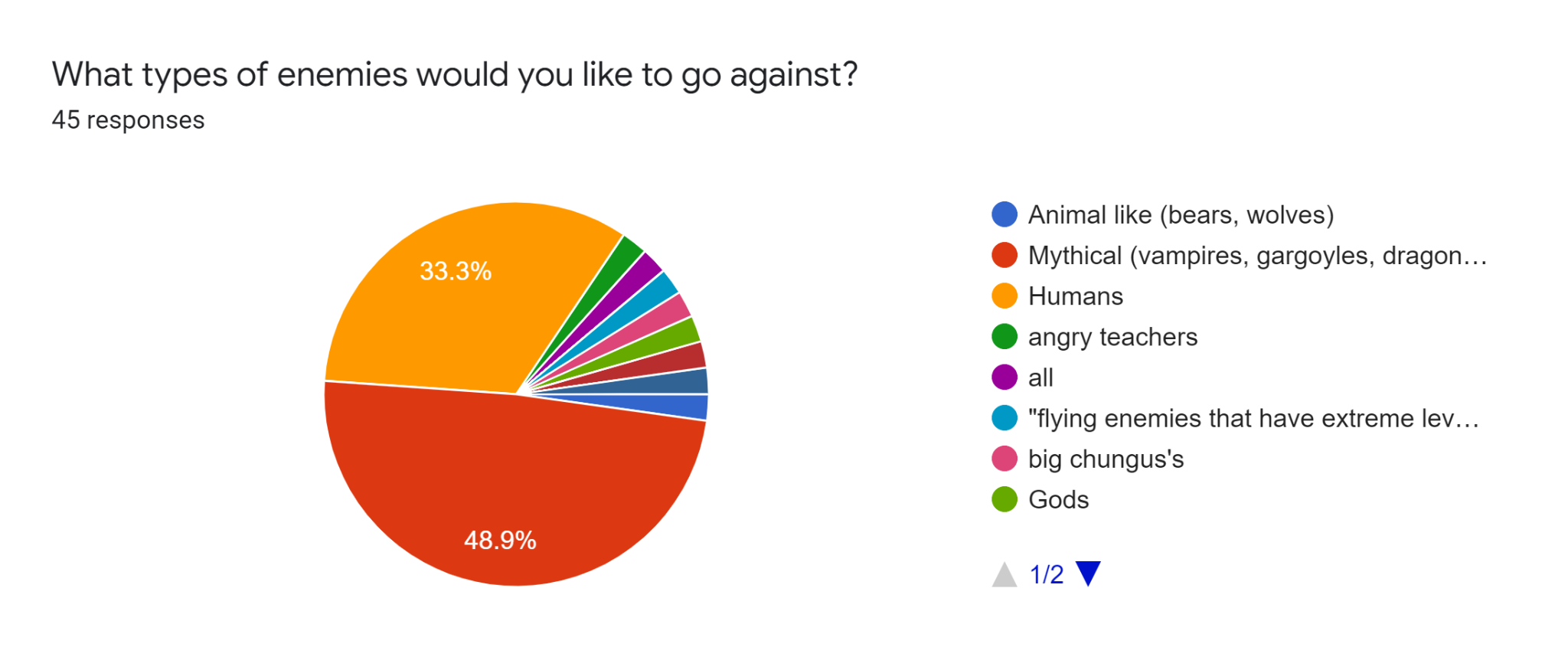
According to the survey, more than half of the respondents would like a game based within some sort of mythical land, such as a flying island or underwater temple. The rest of the participants indicated their interest in a mix of the rest of the options, with the biggest three being biome, inside building and outside building. My main aesthetic theme throughout the game will therefore be a mythical island in which I could incorporate a little of the other 3 options.



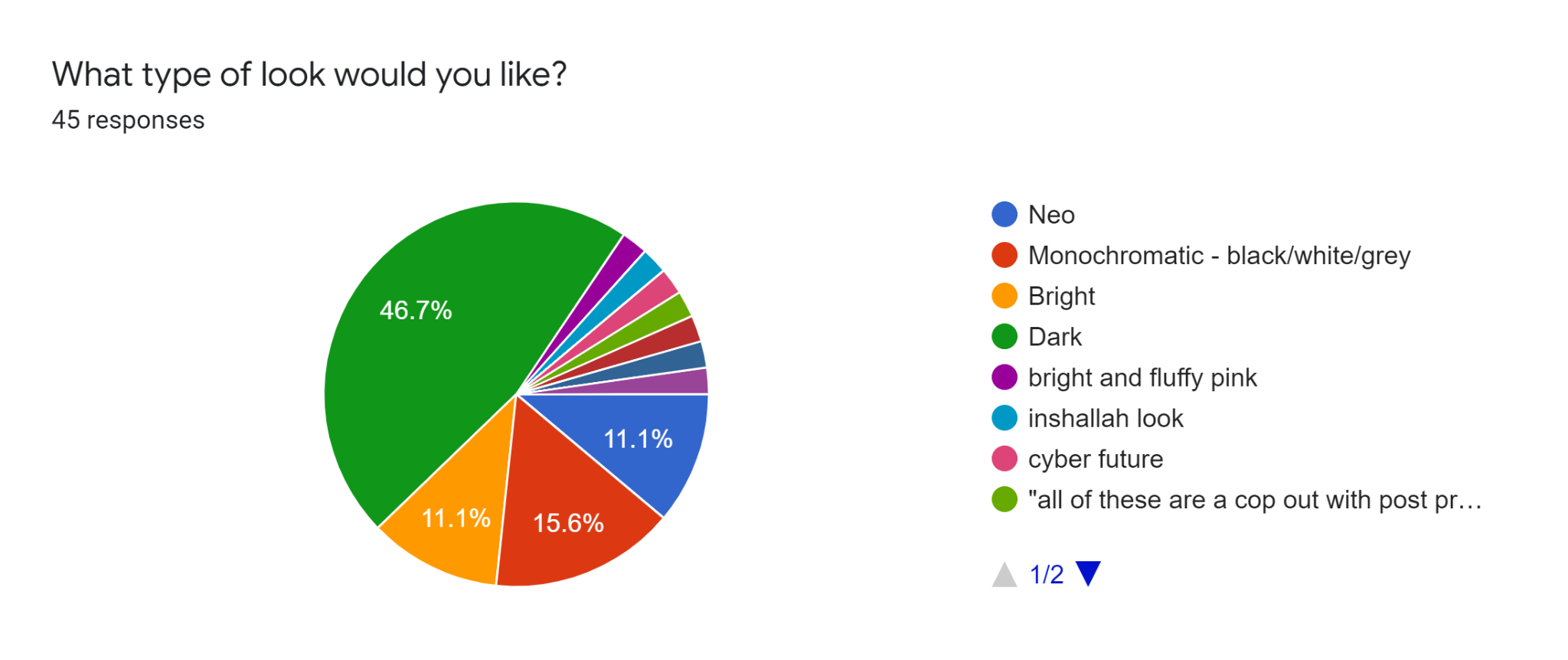
Although melee and simple firearms came very close to each other, melee won overall - this is because many who chose the ‘other’ option said they would like a certain type of melee weapon.



Supernatural powers had a lot of varied responses, with all of the above standing at the top. I have decided to go with no supernatural powers throughout the game, because it’s come to my attention that the game would be reverting itself back into the same generic games being made these days if I did so. Many of the end users I consulted with about why they chose a certain power replied by saying they enjoyed that superpower before in another game, or that they simply don’t think the game will be fun without them. The point of my project is to introduce them to this idea and therefore I have decided not to include any supernatural powers whatsoever.



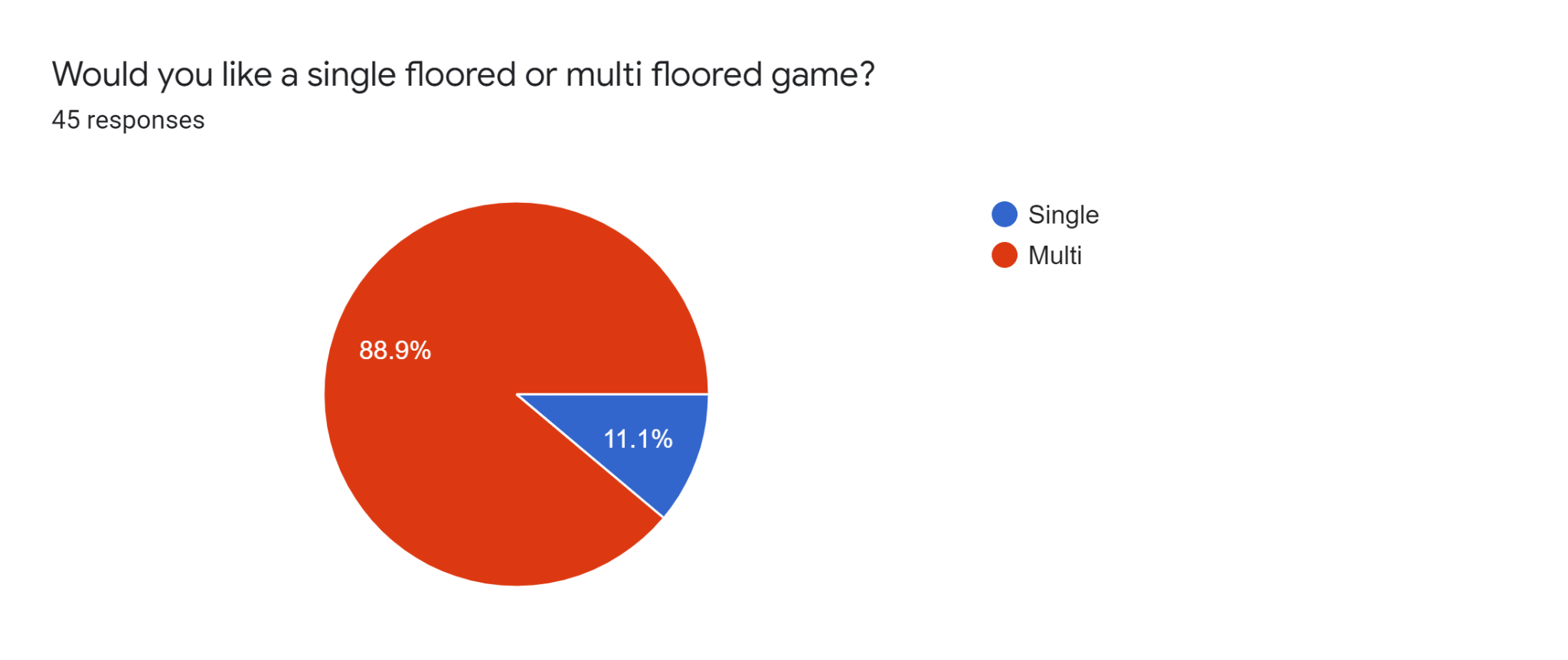
Mythical creatures were picked by half the participants and therefore I will make my enemies mythical creatures and not animal like creatures or human type creatures.



Dark was overwhelmingly chosen as the theme of the game, I don’t see any need to try and include neo as it conflicts too much with the dark theme and would take too much work. On the other hand, I could mix both dark, bright and monochromatic to make a very unique looking game, but this will be quite difficult and I cannot really think of a practical solution to this yet.

I have now thought of an idea that could work, which is the switching of the colour of the sprite of the character depending on the area of the map, or switching between day/night cycles where the map itself changes to the opposite colours. Both these are additional elements but are not essential parts of the game which I might include depending on my end-user.

My end users didn’t think it was a good idea to have what they called ‘conflicting elements’ in the game, and therefore I have opted to not include all 3 elements but go with the type of look that overwhelmingly won - dark.



No question about it, the game will be a multi-floored game, meaning there will be some sort of elevator or next level system. To make the game a little unique to those before it, I have decided to alter the elevator system in such a way that you descend instead of climb up, this will fit with the main storyline of the game in some way, though this will only be learnt through the dialogue. Otherwise, I have decided to create a teleporter which will handle the sending of the player to another level.

## Objectives

*These objectives are not made in any specific order.*

First objective: map and character design. This design needs to fit in with the responses from the survey and include the multi-levelled system that 93.8% of the participants preferred over the single-levelled system. While designing this I must regularly consult my classmates/friends so that I can construct a design that satisfies my end users as much as possible. It should have a mythical land setting with a dark (graphical) theme and multiple floors which the user can enter. The next floor will only be accessible one way through the teleporter that I will add. To progress through the game/map, you will have to kill the enemies in a certain area. A tree structure will be used here where the map is the root node, and the areas are the child nodes. Each area will have sub areas, and each of these will have a corresponding crate game object linked to it. When the enemies in the area are cleared, the list containing the enemies will be checked and if it returns true then the corresponding crate will disappear and the collider attached to it will be disabled.

Second objective: character movement/sound system. This will allow the user to move their character with the WASD configuration. I will also create a custom sound system that measures how much sound the character is making throughout the game and alerts the enemies when too much sound is made. This script will interact with the enemy script and the enemies will be attracted not to the player but to the area where the sound was made. This will be the first objective I complete as without it the game will not be playable no matter what happens.

Third objective: enemy AI mechanism. This will govern how the enemies attack and react in the game. This system should work with the sound movement system, in that the enemies should be attracted to the area the player is in when he makes sound, and that they will deal damage when colliding against him. I will create my own pathfinding method for this that will use raycast(boxcast) to check for collisions against objects and the player. This is the second objective on the list to complete as this also needs to be added alongside the player so that I can start consulting with my end user early on to adhere to their preferences - such as whether to attract the enemy to the player when close enough or still base it on sound made.

Fourth objective: damage and health. The health script will be shared between everything that has health, and there will be no inheritance from this script. This script should just assign a health value to all the game objects with that script attached to them. The damage script will be different for the player as they have a backstab multiplier which means that they will deal more damage when they backstab an enemy. This specific class will use the dot product to find out whether the weapon in the player's hand is facing the back of the enemy. The enemy damage script will be the same for all enemies as they will all deal damage to the player and reduce his health in the same way. Without these components in the game it would be hard to change the level of the game using only the amount of noise the player is making when they are moving, and therefore this should be in the game following the addition of the enemies.

Fifth objective: a pause option and save function for the high scores. They will be saved in a text file and the save function will be called whenever the user quits the game. A merge sort will be used to sort the scores in the text file. The current scores in the text file will be compared against the score saved using the merge sort and the score will be placed in its corresponding position in the list. As only 5 scores will be saved, the score that is then in the 0th element in the list will be removed from the list and the final list is what I will write to the text file. The text file will simply be called highscores. The high scores will be displayed to the user through the NPC character. I decided to do this after coming up with the idea and discussing it with my end user, who said that it was ‘a smart idea’ and they ‘like the implementation of the NPC in the unique way’ that I’ve thought of. This objective will be completed in 2 parts, the first pause option will be completed right after the first objective is complete, while the second high scores save function should be completed right after the health and damage scripts have been finished and incorporated into the game. I’ve chosen this as the high score is partly based on the backStabMultiplier which is part of the 4th objective.

## Pillars

* Look and feel:
  + This will be entirely based on the questionnaire, meaning that a dark theme will be running throughout. The weapon will be a melee weapon, and it will be bright to contrast against the colour of the background and make it easy for the user to see the weapon. The enemies will be red monsters that are blind, this means that the player is more likely to complete and finish the game if they take care when moving and try to sneak up to the enemies to backstab them from behind.
  + The map will be a simple dungeon with many areas, but the user will not have much choice as to which area to go to due to the tree structure of the areas and the way I want this game to be played. New areas will open up once the enemies in that area have been killed, and crates are what will be blocking the enemy from moving further on.
* Interactibility:
  + There will be chests throughout the game that will reward the character with prescriptions. These prescriptions (multiplied by the number of backstabs) will generate the current score of the user. The prescriptions will be randomly generated for each chest.

## Prototypes

*The following is code that was used with a small game with a box that moves around an empty scene.*

CameraFollow.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class CameraFollow : MonoBehaviour {  public Transform player;  public float lagSpeed = 5f;   private void Start()  {  player = GameObject.Find("Player").transform;  }   private void LateUpdate()  {  // Move the camera towards the player's position with a slight lag  Vector3 targetPosition = new Vector3(player.position.x, player.position.y, -5);  transform.position = Vector3.Lerp(transform.position, targetPosition, Time.deltaTime \* lagSpeed);  } } |
| --- |

FirstPlayerMovement.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class FirstPlayerMovement : MonoBehaviour {  public float speed = 1f;   private Vector3 moveDirection = Vector3.zero;   void Update()  {  float horizontalInput = Input.GetAxisRaw("Horizontal");  float verticalInput = Input.GetAxisRaw("Vertical");   moveDirection = new Vector3(horizontalInput, verticalInput);  moveDirection \*= speed;   transform.position += moveDirection \* Time.deltaTime;  } } |
| --- |

# Documented Design

## Overview

The actual game itself will be (according to the survey) inside a mythical land and this theme will be concurrent throughout the full game. The dark theme is a must, and the sprite needs to blend in with the theme as well.

The map will use a tree structure, as said before, and will have game objects called crates attached to the leaf nodes (sub areas) . The root node will be the Map, and the child nodes of this will be the Area nodes.

The weapon will have a collider which checks for collisions against enemies, and when left click is pressed the attack will commence. There will be a cooldown till the next attack, to prevent the player from spamming attacks. The dot product will be used to check whether the player is attacking the enemy from behind and if so more damage will be done.

The high scores will be saved in a text file, and as explained in the Analysis a merge sort will be used to sort the high scores.

There will be a tutorial chest that will display a small message telling the player to collect the chests to gain prescriptions.

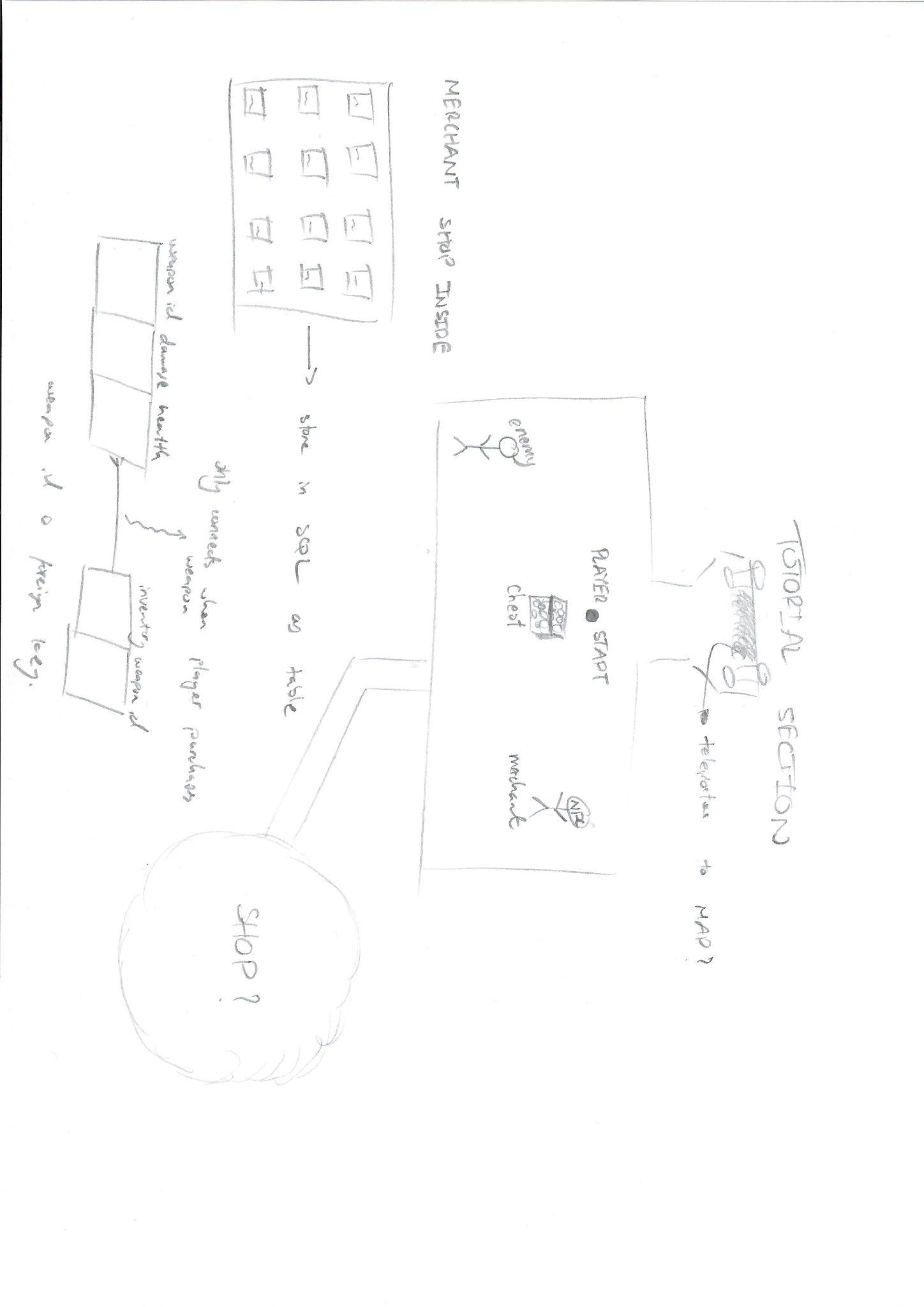
The messages/text will appear on screen for certain amounts of time. This will all be managed by the text to screen manager.

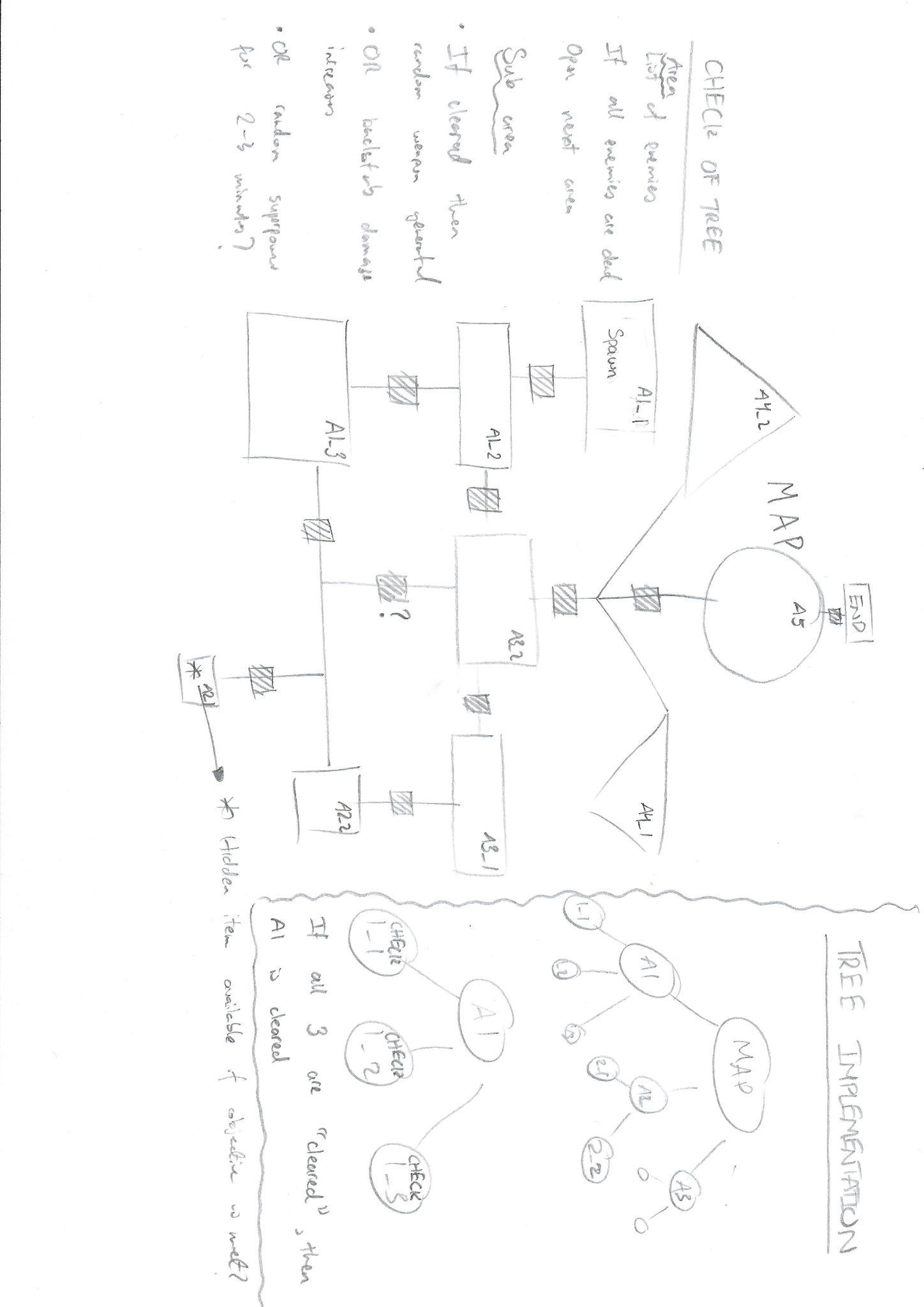
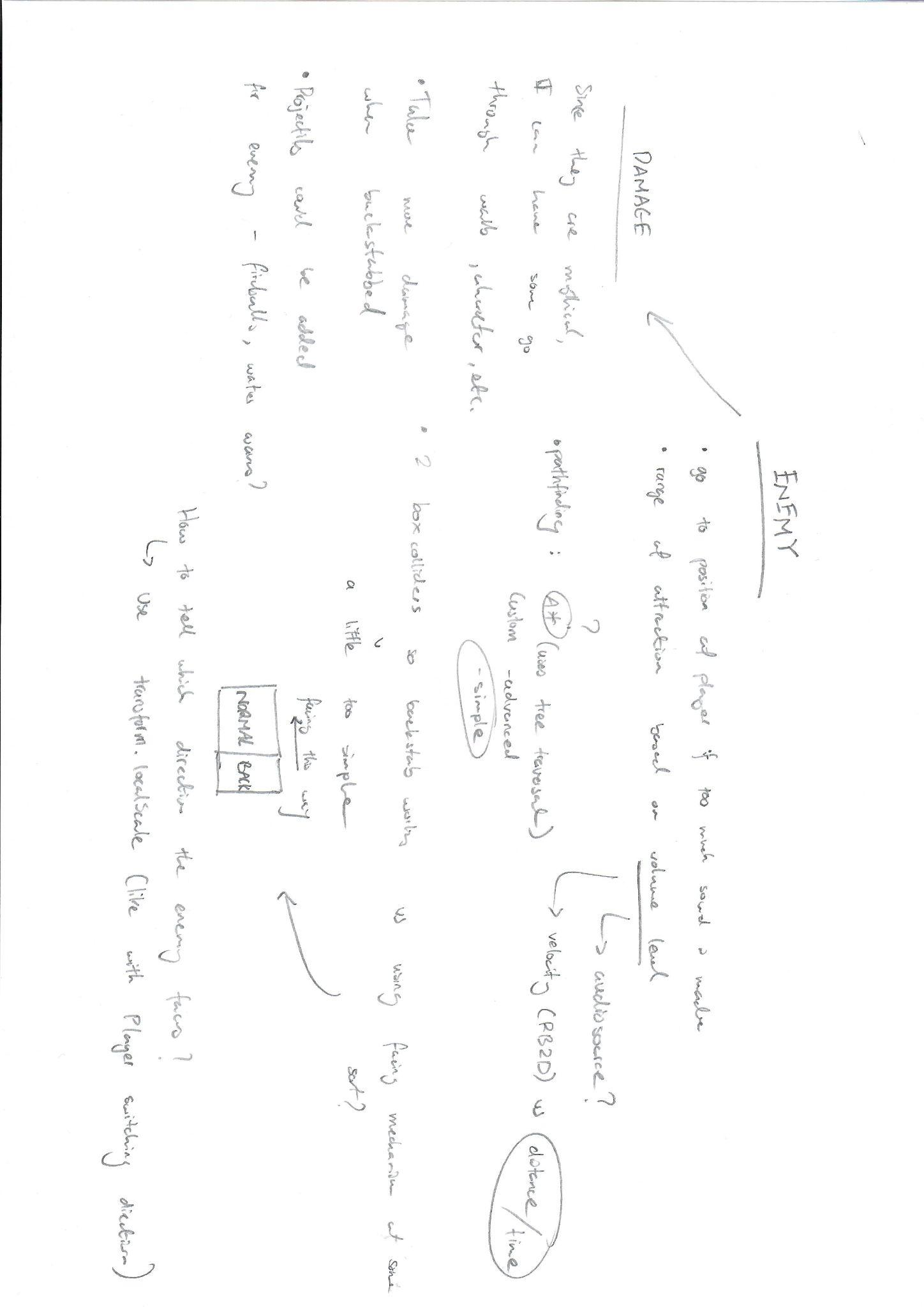
The Gamemanager script will handle the saving and loading of high scores, the increasing of prescriptions, and the text appearing on the screen.

## Notes/Drafts

Things that need to be designed:

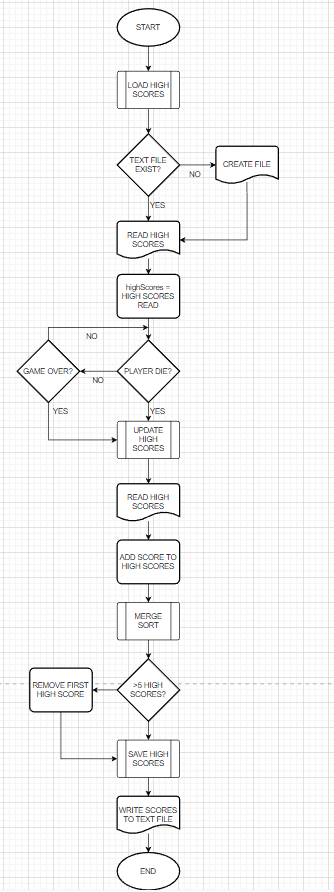
* Entities need to load up when option from Play is chosen
  + Characters
    - Player
    - Enemies
  + Weapon
  + Map
    - Areas and sub areas
    - Crates
  + Audio
    - Audio for sound movement system



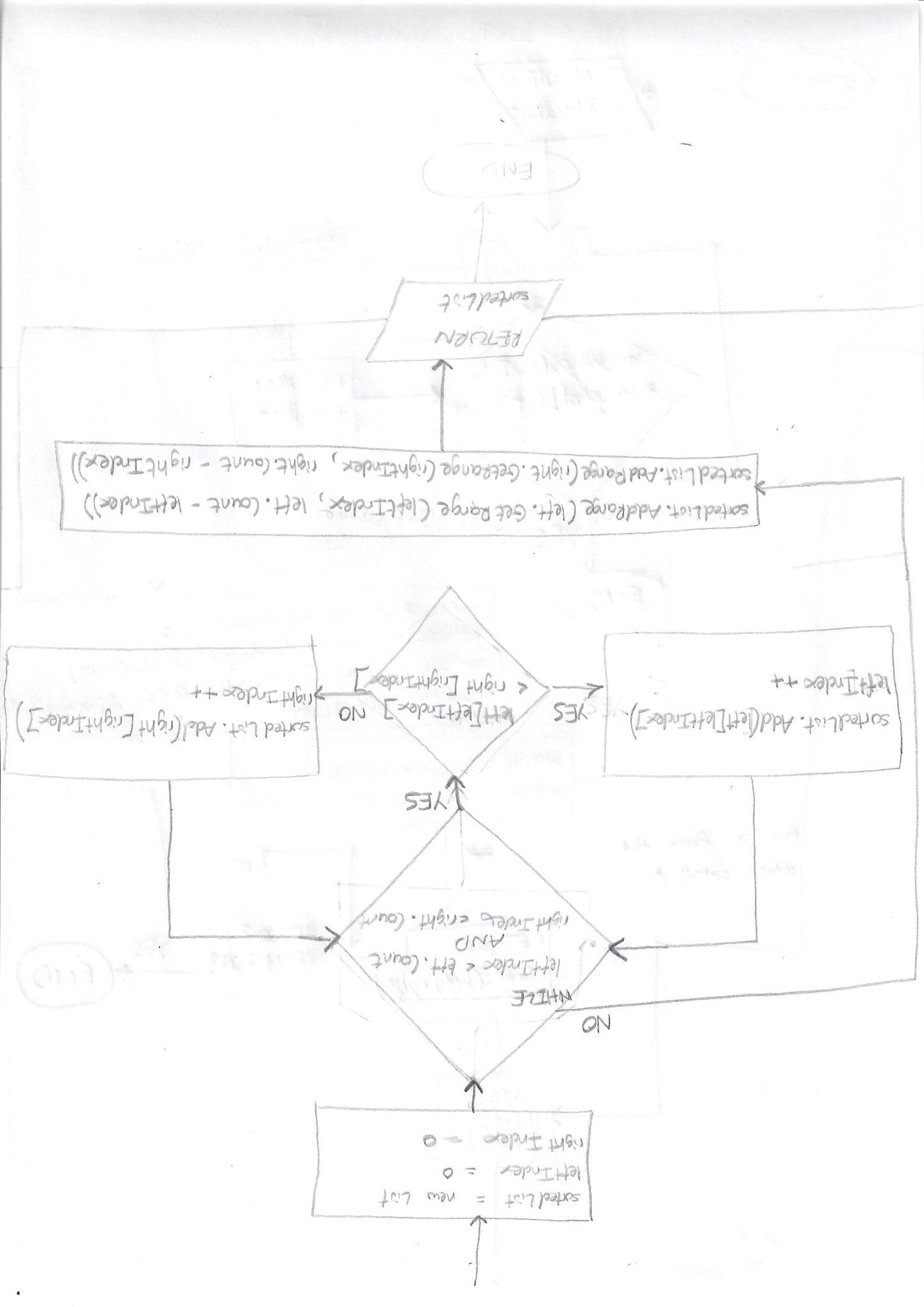
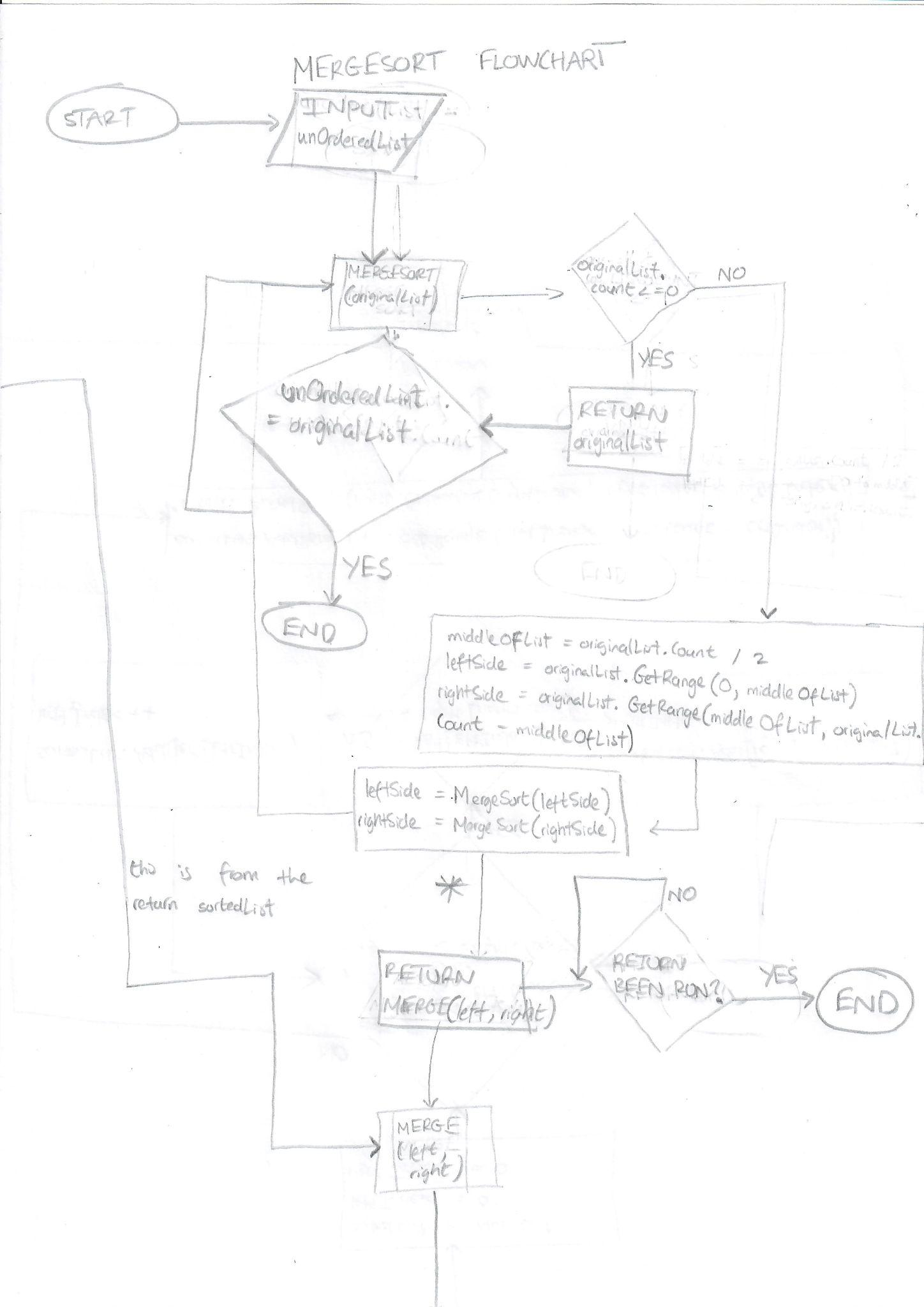


## Flowcharts

High Scores



Merge Sort  
*This flowchart represents the way the game will do this as close as possible, nevertheless some extra statements/variables were required for this to actually work. Furthermore, there is an assumption that the leftSide = MergeSort(leftSide) and vice versa bit does not move downwards before the recursion is over.*

**

## Player Sprite

*All the assets are referenced in the References section of this document.*

This is the first sprite that will be incorporated into the game and will be the one the users see the most. The sprite should fit in and blend in with the map but also be contrasting enough for the user to see the character. The sprite will be flipped on the y-axis when they turn so that they are always looking in the direction that they are moving. This is done using transform.localScale and is seen The character will also have a weapon in their hand and this will move alongside the character.

First Sprite

**

This is the first sprite I will be using as the main character in the game. This sprite will only be used in the beginning of development. There will need to be code to orient the sprite left or right depending on the input (a or d etc.). The sprite will however be required to have 8 directional movement and a collision system so that tests can be run with the sprite and any errors in the code can be amended for when the final sprite is to be used.

Second/Final Sprite



This is the final sprite I will be using in the game. This sprite will be easy to implement and use a box collider with and will allow me to reuse the preexisting code I have to orient the sprite based on input.

## Data Structures

List

Lists are dynamic data structures, which means that they can grow or shrink in size throughout the program. Lists contain methods to add, remove, and access elements at indexes specified by the user. They are much more flexible than arrays as they do not have fixed sizes, but they take up more memory space than arrays and it takes longer on average to access data from a list compared to arrays.

Tree

*Trees themselves are not built-in data structures in C#/Unity but they can be implemented by defining classes for them and/or using data structures such as lists and dictionaries.*

Trees are non-linear data structures, as opposed to Lists which are linear data structures. The difference between these two types of structures is that linear data structures are data structures where the data elements are arranged sequentially or linearly whereas non-linear data structures are the opposite.

A tree specifically refers to a hierarchical structure between data elements that are called nodes. Characteristics of trees include:

* Each tree has one node called the root node and this node does not have a parent.
* Every node other than the root node has a parent node.
* Each node is connected to its parent node/children node by edges.

The tree in my game is used to represent the map. The root node is the GameObject with the MapManager.cs script attached to it, and all the nodes of the map are connected via the Area.cs script attached to them.

I have used a tree for the map as I can efficiently search, insert, and delete areas of the map very easily and quickly when I need to do so. Using a tree also means that when it comes to checking whether an area has been cleared I can quickly/efficiently do so through traversing the tree through a recursive algorithm, and that I do not need to represent the map with a collection of many data structures bundled together.

Overall, the tree data structure is a very useful structure that can be used to save space as it can represent complex hierarchical relationships in more compact ways and gives the user ease of traversal for checking for conditions by using recursive algorithms.

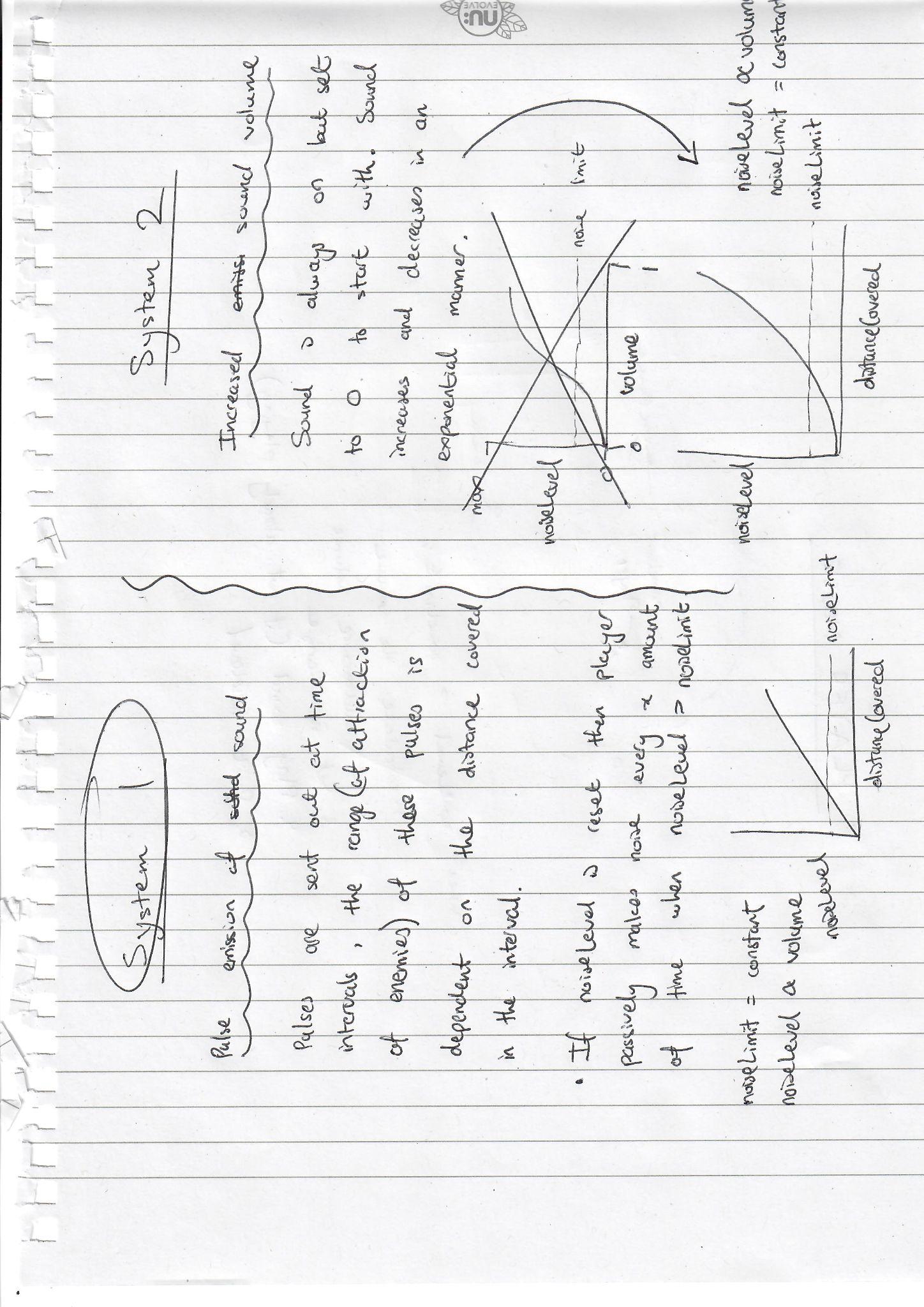
## Pseudocode, Algorithms, & Mathematical operations

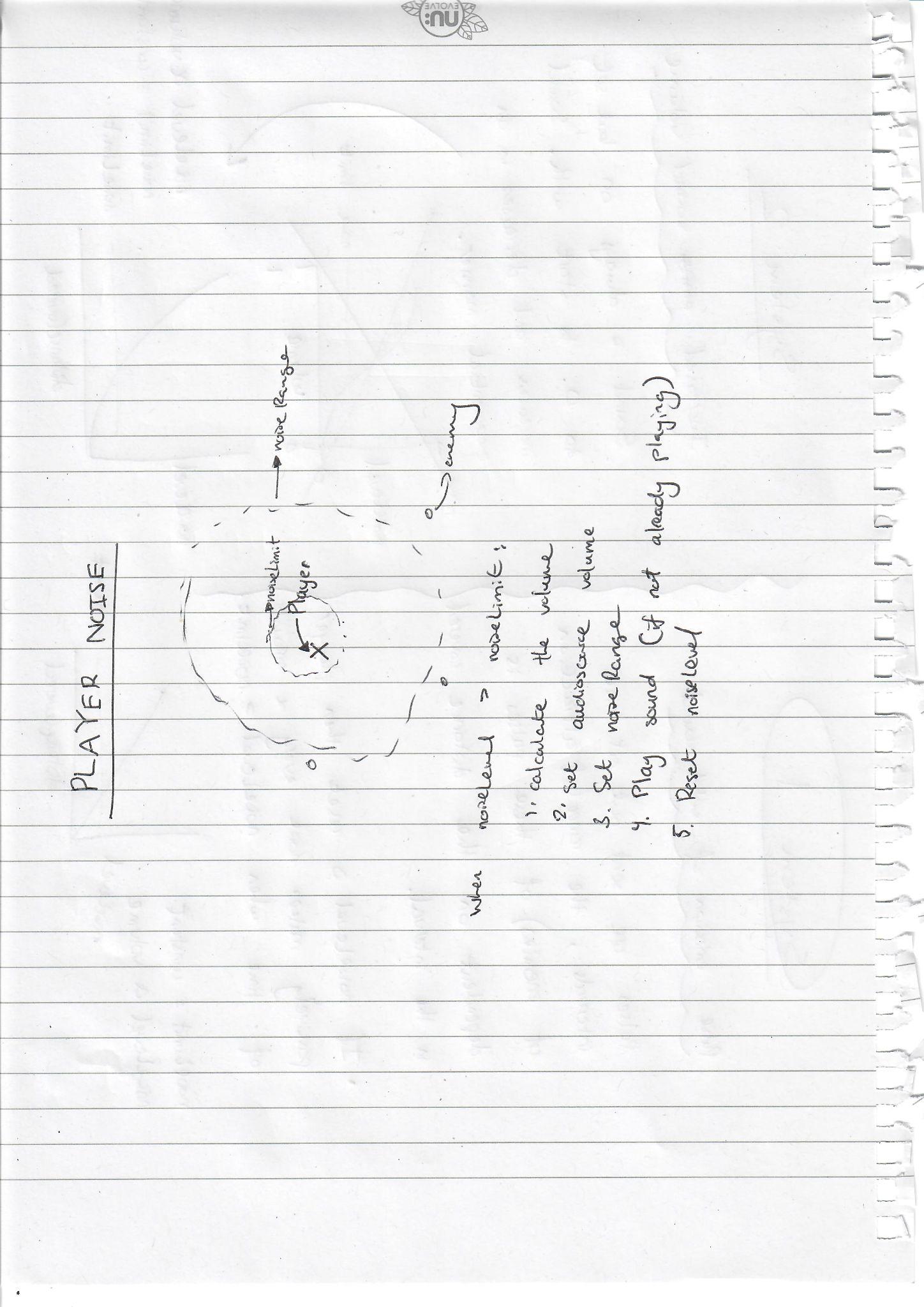
Player Noise System

The essence of this game is rooted in this algorithm, which provides the framework for the rules, behaviours, mechanics, and overall gameplay experience.

Instead of relying on the velocity of the Player through RigidBody2D to determine the volume of the sound, I have opted to specify the distance and time parameters myself. This gives me the flexibility to mix and match the values according to my end user until I achieve the balance that they want.

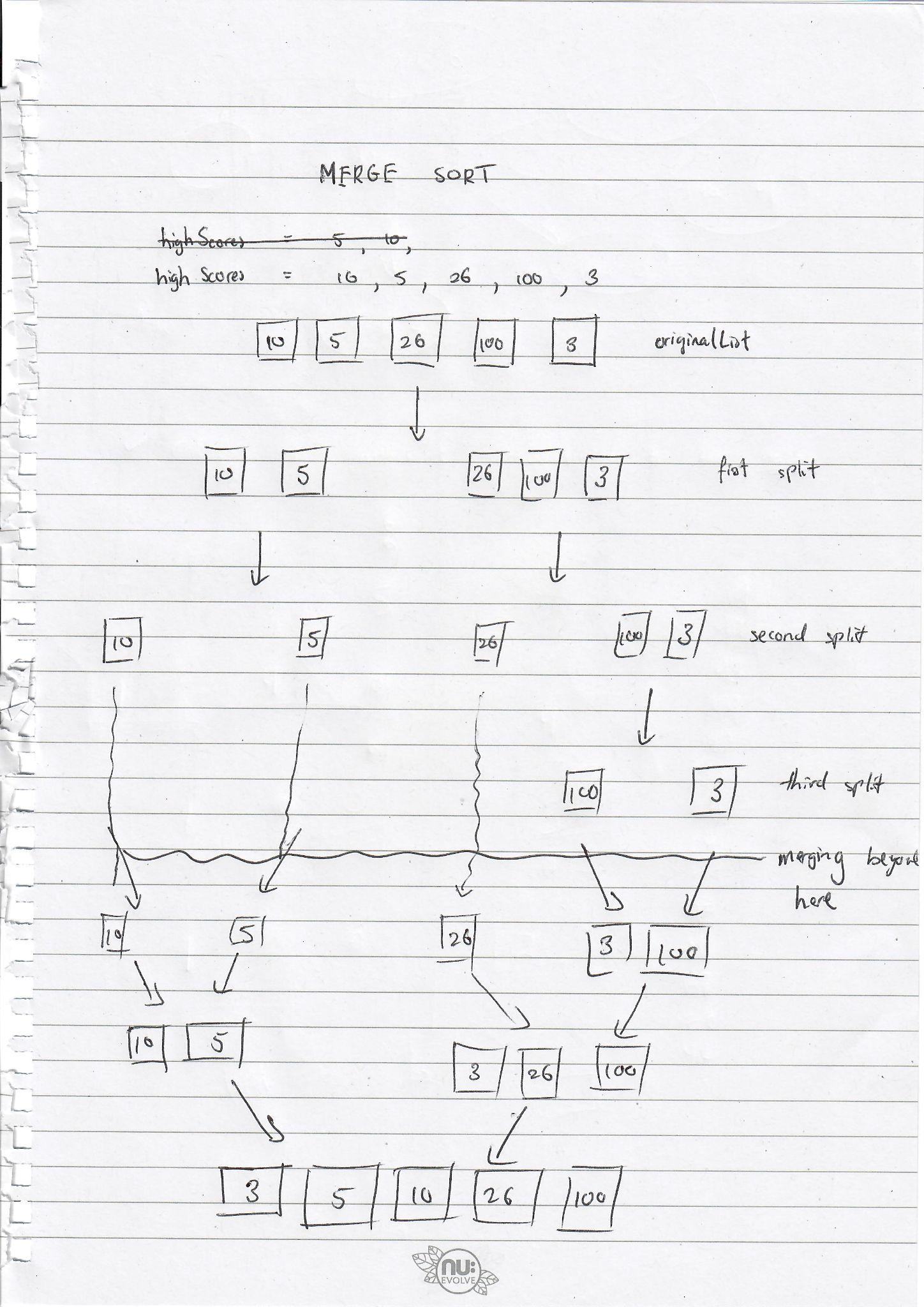
There are 2 main systems that I have in mind for the Player Movement, System 1 that used a pulse method of emitting the sound, and System 2 was just an increased volume of sound that was effectively always playing. System 2’s noiseLevel was different from System 1’s due to its exponential relationship with distanceCovered as opposed to System 1’s noiseLevel’s proportional relationship with distanceCovered. I’ve decided to go ahead with System 1, but both relationships and the concepts of the different systems can be seen below.

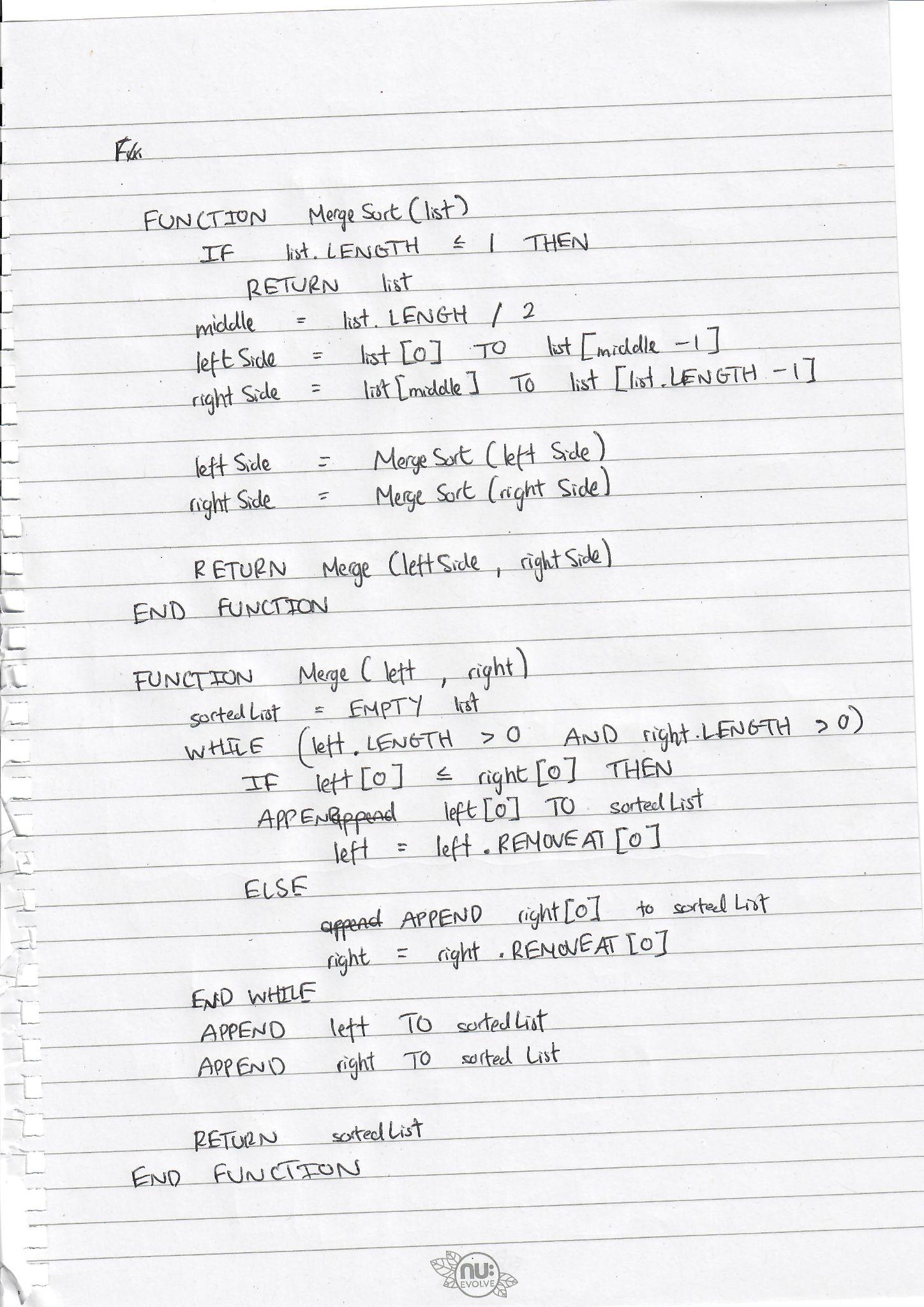


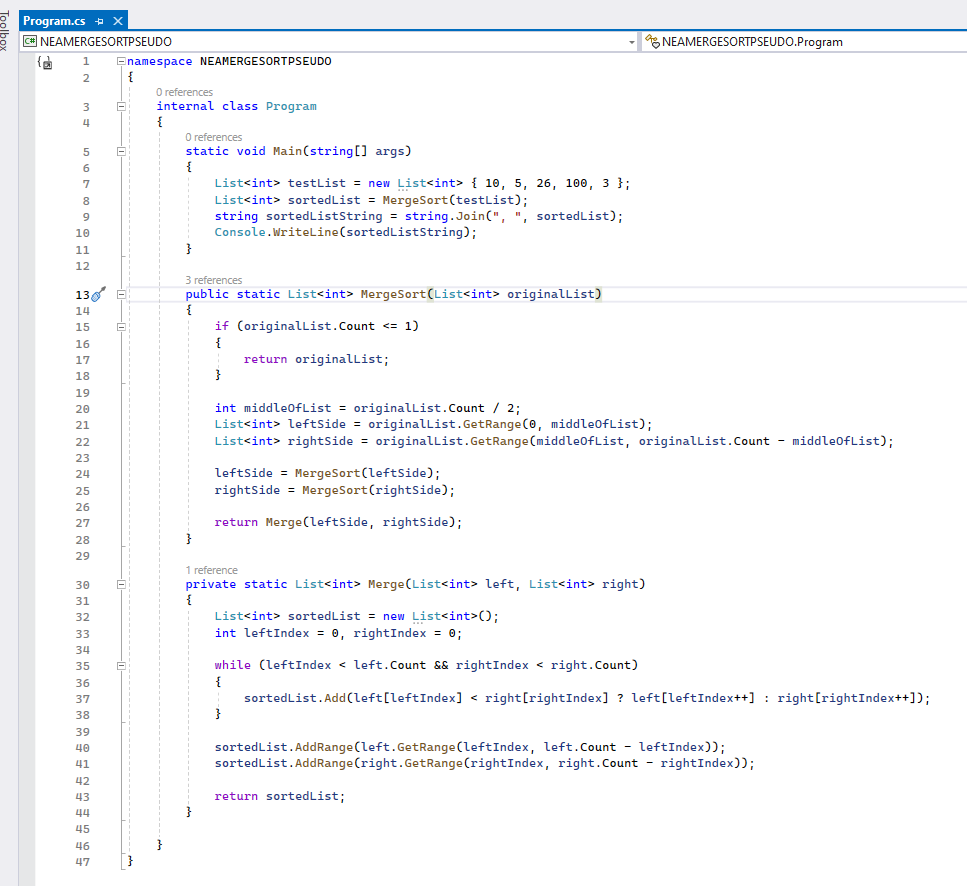


Merge Sort

This is a sorting algorithm and will take in the high scores list and recursively divide it into 2 halves until each element is in its own sublist. The algorithm then sorts each of the sublists and merges them back together where they are now in order. The output list will be the original list sorted in ascending order. A visual representation of the sorting algorithm can be seen below along with the pseudocode. I also created a prototype of the code in visual studio as I wanted to test the ternary operator and how it can be used within my code.







*I will provide an explanation of how the ternary operator works underneath. The statement using it is on line 37.*

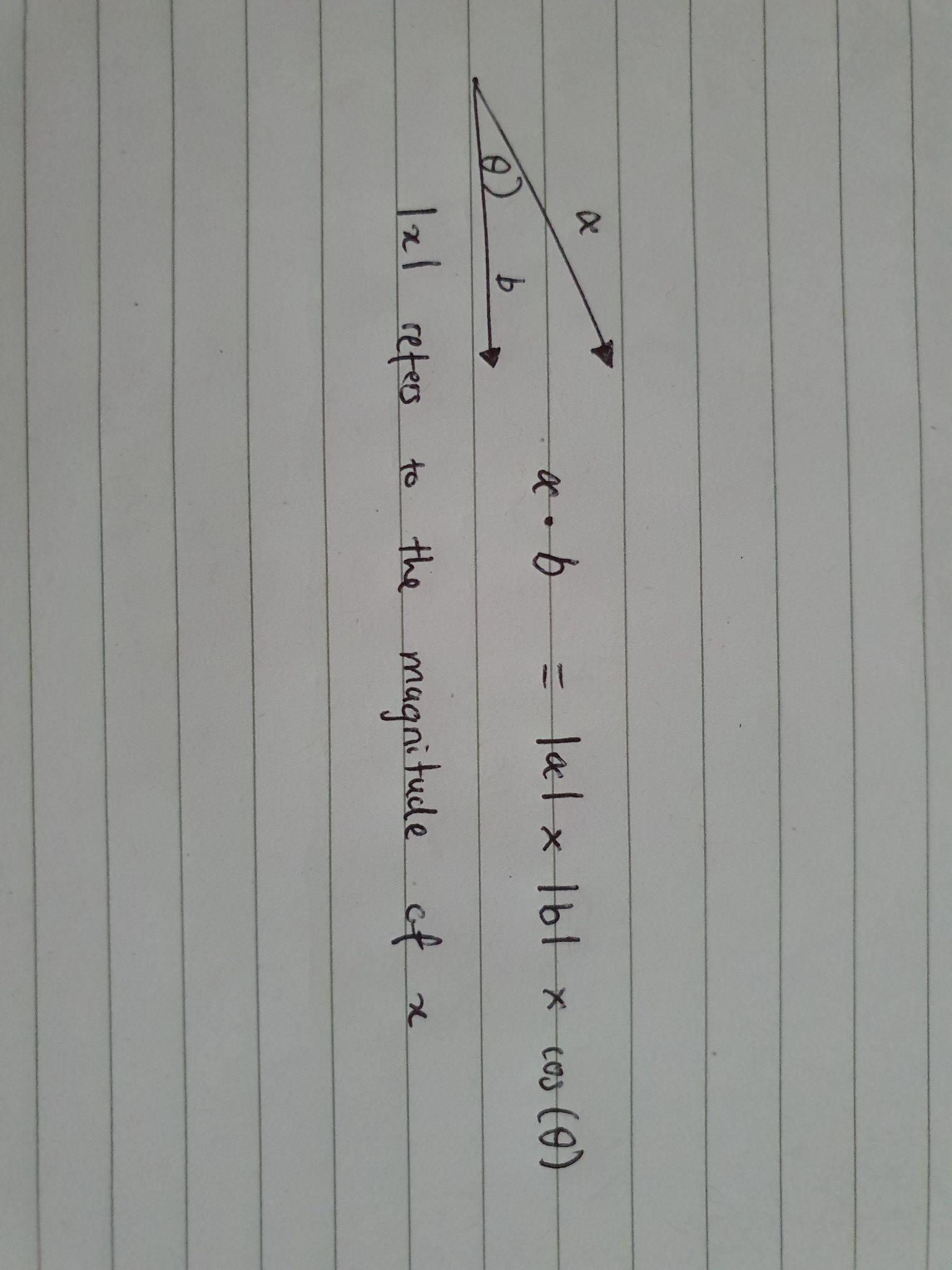
The ternary operator is a shorthand way of writing an if and else statement in a single line of code. It is represented by a ? and works with a boolean condition at the start. If the boolean condition evaluates to true then the first statement after the ? is the statement that will be run, otherwise the second statement which is after the : is run.

You can see the ternary operator above on line 37. In my case, the condition is ‘left[leftIndex] < right[rightIndex]’. If this condition is true, the value of ‘left[leftIndex]’ is returned and added to the list and then left[leftIndex] is incremented by 1. Otherwise the same happens but for the right[rightIndex]. The order of operations here and the increment specifically are very important to make sure that the value of left[leftIndex] is used in the sorted list and then the value of left[leftIndex] is incremented.

Dot product

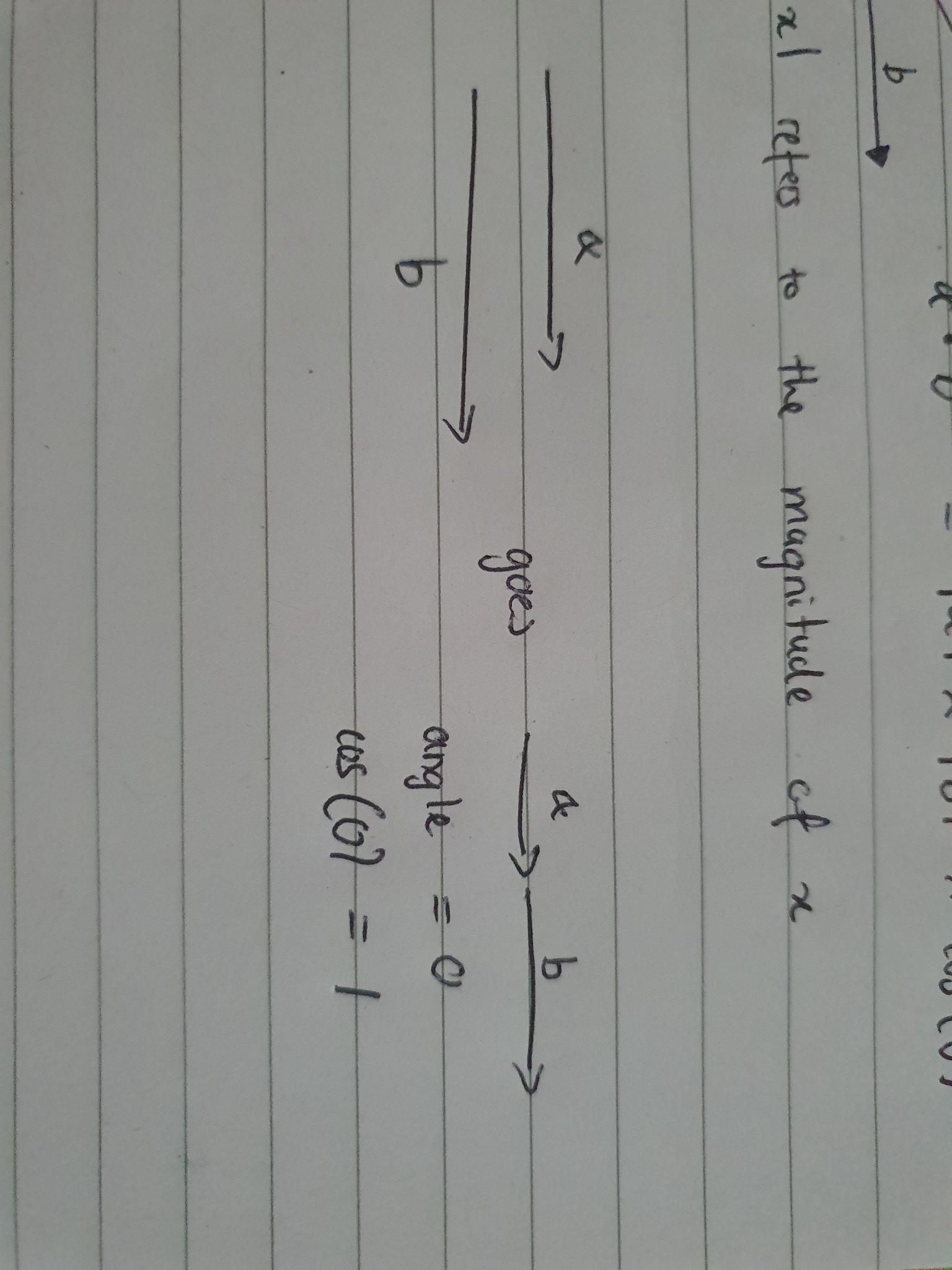
The dot product is an essential part of the player damage mechanism as this is a fundamental part of the calculation of if the player is backstabbing or not. This particular calculation can be found in Weapon.cs script at the end of the script.

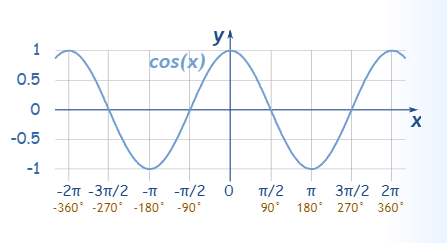
The dot product is a mathematical operation that takes 2 vectors as input and returns a value. This value is a scalar. It is calculated like this:



In my game, the dot product is used to determine whether the player's attack is a backstab or not. I will do this by using the direction of the weapon and the direction from the weapon to the enemy as my 2 vectors. To do so, I will access the direction of the player’s weapon through its transform.right property, and the direction from the weapon to the enemy by subtracting the enemy’s current position from the weapon’s current position which will result in a vector that points from the weapon towards the enemy. So the 2 vectors are ready now.

To go on to understand how the rest works, I need to go over the dot product in further detail and the meaning of the cosine of the angle that is between the 2 vectors. So, when the dot product is calculated, it returns a scalar value that is equal to the product of the magnitude of the 2 vectors multiplied by the cosine of the angle between the vectors. Now, if the 2 vectors are parallel like below:





[*https://www.mathsisfun.com/algebra/trig-sin-cos-tan-graphs.html*](https://www.mathsisfun.com/algebra/trig-sin-cos-tan-graphs.html) *- 4:29 - 5th March 2023. Only take a look at the positive part of this graph, the right bit.*

Then the cosine of the angle between them will equal 1 as seen on the graph. This means that when parallel vectors are used as the input vectors for the dot product, the dot product is equal to the product of their magnitudes.

What about perpendicular then? Using the same logic as before we come to the conclusion that the cosine of the angle will equal 0, therefore the dot product will equal 0. If the 2 vectors are parallel but in opposite direction, the cosine of the angle between them will be -1 and the dot product will be equal to the negative product of their magnitudes.

Now that we have the cases and understand the values that might pop up from the dot product calculations, we will need to reverse our thinking a little. We can take a look at the if statement I will have in my code as simple English just to wrap our heads around it.

If the dot product is less than 0

// Cosine of the angle between the vectors is negative

// Look at the graph again

// Cosine of angle is negative when 90 < angle < 270

// In our context, the weapon is behind the enemy!

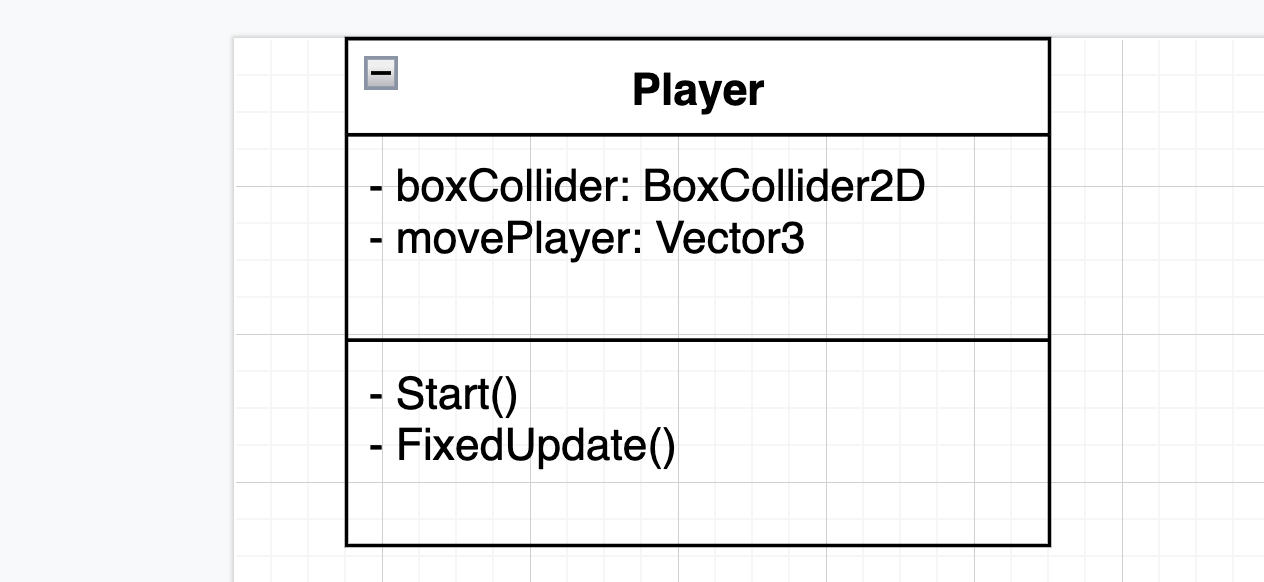
It took me quite some time and research to understand how actually the dot product can be used alongside the rest of the code and what type of variables I would need to let the calculation work as it should. The assumption we made in the first comment only works due to the fact that I will have a checker to set the vector of the weapon to the enemy to a flipped version of itself if the enemy is facing right. This is just to ensure that the positive or negative value of the dot product I get represents the positive or negative value of the cosine of the angle. Otherwise we would not be able to make this assumption since if the vectors are not pointing in the same ‘plane’ then it would be difficult to know whether the value of the dot product is being represented by the cosine of the angles positive/negative value.

## UML Class Diagrams - Individual

*All diagrams with R after a dash (Script Name - R) are scripts that have been copied from the internet and the reference to the creator of them can be found in the appendix section of this document.*

Player - R

*This script has been replaced by PlayerMovement.*

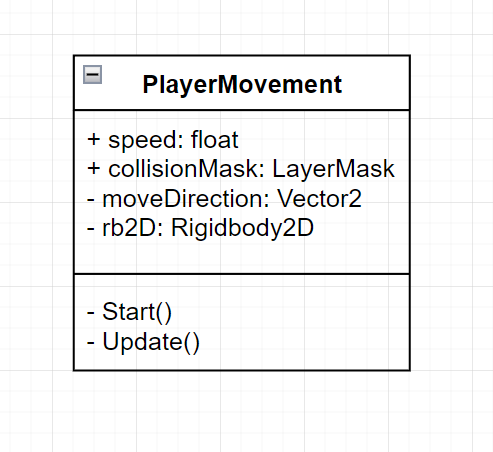


The in-game player class diagram will initially look like this, with fields boxCollider and movePlayer, and methods Start and FixedUpdate. These fields and methods will make the in-game character move and have a collision detection system which will prevent the character from phasing through objects/characters/sprites. The character will be controlled through both the WASD and arrow keys so that no inconvenience is brought upon any of the end-users who prefer to use one over the other. FixedUpdate is used in this case as it updates at regular intervals and after any fixed update is called all necessary physics calculations are made immediately - making it much more suitable for rigidbodies than Update.

Player Movement

*This script is my very own custom-made script and replaces the Player script.*

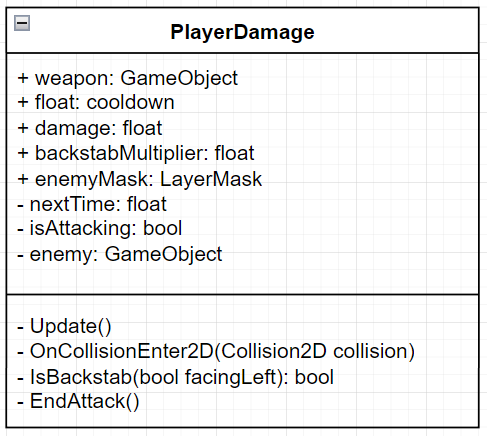
Through testing the player script and coding it, I realised I could create a more efficient and practical script for the player and his movement/collision detection against walls, etc. This script will also allow me to debug the problems I had before with the player phasing through certain areas and going halfway through the walls (this is described in the Testing section). The diagram for this script can be seen below:



As seen, this script differs a lot from the Player script. Here, I have decided to use to my advantage the RigidBody2D class of the unity engine, which will allow me to add realistic physics properties to my player (and other game objects). However, I decided that I still want to stick with my own script that checks for collisions as opposed to using the RigidBody2D scripts available to me, which can be seen within the technical solution. The attribute collisionMask which is of type LayerMask is a public attribute that will give me the ability to assign specific layers to it. These layers are the layers that will be checked against for the collision detection using BoxCast. The RigidBody2D will only be used to set the velocity of the player for now.

Player Damage - 1

*This script was updated to Player Damage - 2.*



PlayerDamage deals with all the damage the player deals. This script works in conjunction with the Health script, which deals more with the actual health being reduced and character dying. The script focuses on whether the player is allowed to attack right now and if so how much damage the player should deal.

The amount of damage that is dealt depends on if the user is backstabbing the enemy or attacking them normally. If they are backstabbing, more damage is dealt to the enemy and the more backstabs the player does the more damage a backstab does later on in the game. Furthermore, the current score of the player is calculated by the number of prescriptions the player has multiplied by the backstabMultiplier variable.

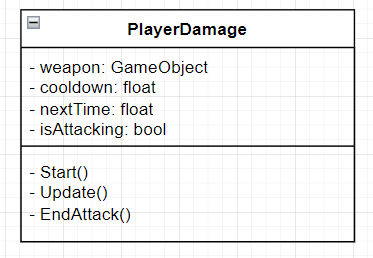
To make sure that the correct enemy is being dealt damage to, when the player collides with an enemy that enemy is set as the enemy variable I declared.

While making this code and looking at the UML class diagram I realised that I made a mistake, and this happened to show up when I was testing the attacking of the player. I therefore had to update this script and create another script to work alongside it. It also makes much more sense to have a weapon script specifically for the weapon, since I actually want to check if the weapon is colliding with the enemy and was accessing the weapon GameObject multiple times in PlayerDamage to check for its position, direction, etc.

Player Damage - 2

*The updated version of Player Damage - 1.*

*This script was updated to Player Damage - 3.*

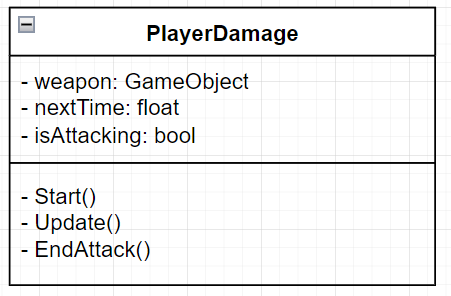


The new and improved PlayerDamage script only deals with the switching on and off of the BoxCollider2D component. This is entirely dependent on the amount of time since the last attack, and it makes sense for this to be on the PlayerDamage script as opposed to the Weapon script since I want this to be the controller of the weapon damage.

This script will calculate the time till the next attack and switch the collider of the weapon on if the cooldown has elapsed. The script will also switch the weapon collider off after the attack has been completed, but needs to give enough time for the weapon class to calculate the amount of damage to be done to the enemy and deal the damage before setting isAttacking to false again and switching off the collider on the weapon.

Player Damage - 3

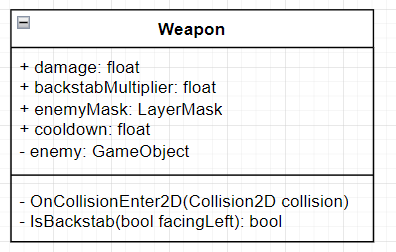
*The updated version of Player Damage - 2.*



An almost imperceptible change, cooldown is no longer an attribute. The fact that I was accessing the Weapon script of my weapon GameObject directly was not correct, and so I changed my Weapon script so that there is a getter method for the cooldown. This will help promote encapsulation and restrict me from exposing the cooldown variable directly.

Weapon - 1

*This script was updated to Weapon - 2.*

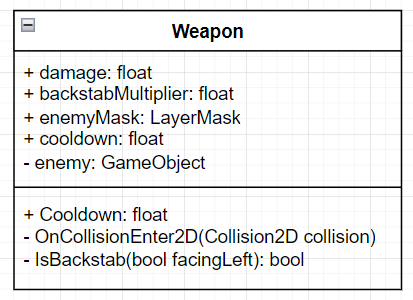


Weapon works with PlayerDamage and is attached to the weapon that the character holds. This script will calculate the amount of damage dealt to the enemy, deal that damage, and contains the cooldown that is set public. The cooldown is the main attribute that is shared between the 2 scripts, but the OnCollisionEnter2D method will be somewhat controlled by the PlayerDamage script. The theory is that this method will only be run when the collider is actually on, as otherwise no collisions can actually take place. Therefore the PlayerDamage script will control whether or not the BoxCollider is actually enabled or not.

I’ve declared cooldown publicly so that it is easy to change through the Unity inspector, but I’ve changed it from being in the PlayerDamage script (as seen in PlayerDamage - 1) to this script so that the cooldown can be changed for a weapon specifically as opposed for all weapons. This gives me greater freedom and customisation if my end user wants me to add more weapons.

Weapon - 2

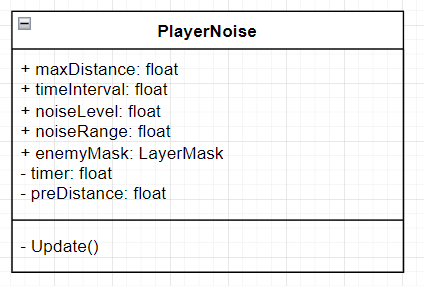
*The updated version of Weapon - 1.*

**

The script was only updated so that there is a getter called Cooldown that gets the value of cooldown so that the PlayerDamage script can use it without directly accessing the cooldown attribute.

Player Noise - 1

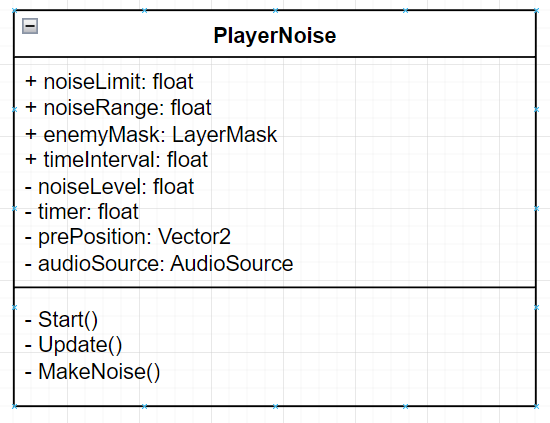
*This script was updated to Player Noise - 2.*



This is my first diagram of the PlayerNoise class. This class didn’t fit the game properly due to the unnecessary number of public attributes that could have been private. Furthermore, I did not include the AudioSource component that will be played when too much noise is made by the player. The updated version is much more realistic and makes use of the AudioSource component.

Player Noise - 2

*The updated version of Player Noise - 1.*



PlayerNoise is one of the most important scripts that I will add into the game. This script represents most of the second objective stated in the Analysis section of this document. It will attract enemies through the MakeNoise() method that will interact with the Enemy script attached to my enemies and call the Attract method that will set the position my player made the noise at as the position the enemies should be attracted to.

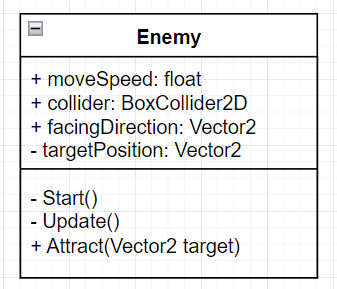
The users will be able to hear the amount of sound they are making as an ominous sound will play throughout the game when they are moving, and the volume of this sound is proportional to the distance the user covers every time frame. The time frame is represented by the timeInterval attribute of the class; I decided to do this instead of using FixedUpdate() or Update() as this doesn’t give me the level of customisation I want to have in my game. It also means that I will be able to alter this value till I achieve a balance between time elapsed and how often the audio is played.

The user does have freedom to move around a little without making sound, which is the whole point of this game. This is directly related to the noiseLimit, and I will alter this value again so that I can achieve a balance between how hard and fun the game is, as if the noiseLimit is too low my users may not enjoy the absurdly slow rate they have to move at. noiseLevel also passively increases a little as long as the player moves, this is to add a little difficulty to the game and ensures that when there are multiple enemies grouped together, even if you move slow, you will still begin attracting them towards your position due to the (near) passive generation of noise.

I’ve also gone one step further and have created a noiseRange, which is the range at which enemies will be attracted to the enemy. This will be calculated based on how great noiseLevel is compared to noiseLimit. This means that enemies aren’t just attracted to the player within a set radius, but that this radius changes alongside how much they are moving around.

Enemy - 1

*This script was updated to Enemy - 2.*

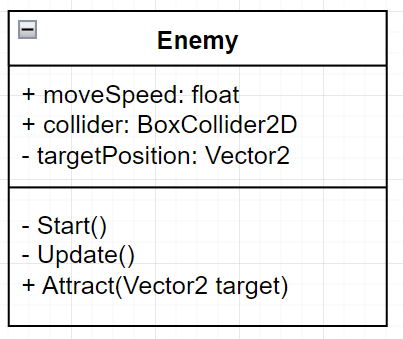


The Enemy script will be attached to every enemy in my game. This will assign them a speed, check for collisions, assign them the direction they are facing in, and a target position to which they are attracted to. The facingDirection will be used when calculating whether the player is backstabbing the enemy or not, and the targetPosition will be where the player was at when the noise was made by them.

PlayerNoise uses the attract function of this script as seen, which will set the targetPosition to the position of the player at that time. Boxcast will be used to check for collisions against walls.

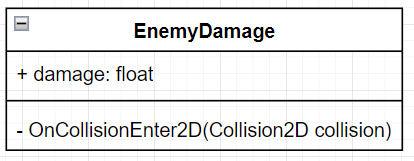
Enemy - 2

*The updated version of Enemy - 1.*



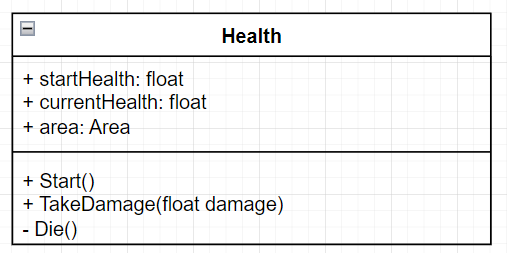
The facingDirection attribute was removed, and the calculation of whether they are facingLeft or not is now done inside the Weapon script instead. This is more efficient and less space consuming then declaring the facingDirection attribute and setting it to a value dependent on the transform.localScale.x value and then accessing that, and this is especially true in my case since my Weapon script has a reference to the enemy GameObject anyway and is capable of accessing the value of transform.localScale.x without any extra functionality.

Enemy Damage



EnemyDamage is a simple script that deals damage to the player when the enemy collides with them. This is done by checking the name of the GameObject that they collided with. The damage will be set to a default value of 1 but can be easily changed for increasing difficulty of enemies as the game progresses and the user nears the end.

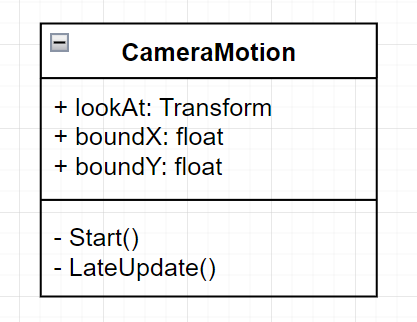
Health



The startHealth refers to the amount of health each object will start with, and the currentHealth is how much it has at any point in time. The reference to the Area script here is because of the fact that when an enemy dies, the enemy should be removed from the list of enemies they belong to which is connected to the area script. To do this, I will iterate through the enemy GameObject name and the Area that it is connected to till I find a match, and then delete that element from the list.

If however the gameobject that died is the player, I will have an if statement that will check this, then the application will quit and the game has ended.

Camera

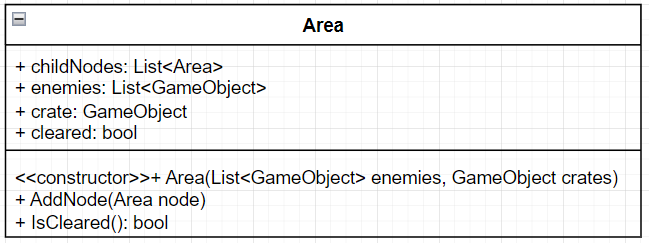


The CameraMotion class is a script that will be added to the main camera within Unity. I will use this class specifically and this layout as it will give me more freedom later on whereas other code which only follows the sprite directly doesn’t have the freedom mine has. Both boundX and boundY are fields which are assigned a value, they are set to public so they can be edited in the Unity inspector and refer to the amount of space the character can move before the camera begins following them. LateUpdate() will be used instead of FixedUpdate or Update as the process of calculating whether the camera needs to move or not needs to happen after the sprite’s movement has occurred.

For testing purposes and simplicity, I will use the normal camera which does not have a delay or bounds whatsoever, it will simply follow the character around while they stay in the middle of the screen. However, for my end-user this will be changed based on what they prefer more after I consult them on it.

Area - 1

*This script was updated to Area - 2.*

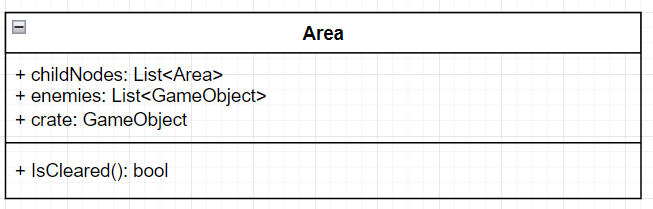


This is most of the code that will be used to create the tree that represents the map. The constructor will take in 2 arguments, enemies and crates, and will ensure that every Area object is created with a list of enemies and a crate. The actual areas will be created by the MapManager script itself, which will deal with which area is linked to which and the creation of each and every node. The boolean variable cleared is set to false initially, and the IsCleared method will check if an area is truly cleared or not. If an area is cleared then cleared will be set to true. IsCleared will be a recursive method that checks the leaves of the node before checking the node itself. If cleared is true, the collider on the crate should be disabled and the crate GameObject should also be hidden. I cannot delete the crate as the crate is what the Area script will be attached to.

This script will also allow for there to be child nodes of each node

Area - 2

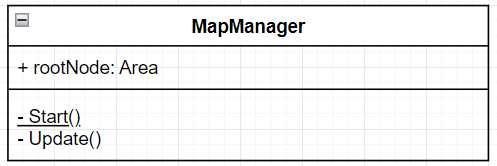
*The updated version of Area - 1.*

**

Area - 2 removes the cleared boolean attribute, the add node method, and the constructor. While creating this script alongside the MapManager, I realised that creating the tree in the MapManager privately was not the way I wanted to go about creating the map. To make it easier to expand the map and include new areas, it would make more sense to create the areas through the Unity inspector and dragging and dropping into public fields. I therefore made the MapManager script nearly entirely useless albeit the numerous lines of code I had written in it. Creation of the areas will now be simpler and more comfortable for me than having to write multiple lines of code for the creation of an area. For a smaller game, it is perhaps more appropriate to create the areas privately within another script, but this is not the case for my game.

Map Manager

*This consists of the old Map Manager and the new one - instead of creating Map Manager - 1 and Map Manager - 2 I included both since this will be easier to understand this way. The underlined bit is the part that was originally in Map Manager but is no longer in the script.*

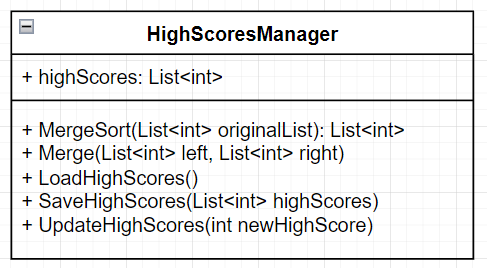
**

The original script created the Map node by node using code and the constructor that was originally part of the Area script. Start() is where all the nodes were created which makes sense since this is when the game starts. However this was inefficient and time consuming which led to the removal of the Start() method and the creation of the map using the Unity inspector.

The new script does this publicly, and the MapManager will only be added to the component that is the root node. This component will use the Update() method to check (every frame) whether it is cleared using the IsCleared method of Area. This method as explained in Area recursively calls itself and checks all nodes that lie under this node. As this node is the root node, this means that every node will be checked.

High Scores Manager

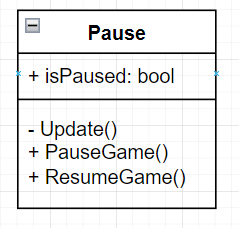
*This class and its methods/attributes are all static.*



I believe that making everything static is the correct way to go about creating this class and all of its methods/attributes as this class does not have any object related code or need to be instantiated at all. The GameManager will act as the controller for this class, this class will only contain the algorithms and methods that I will be using through the GameManager. Thanks to this class being static, it will be easy to use the methods of this class in the GameManager and it will be easier to manage and maintain the high scores. This will also help with encapsulation since it will be impossible to accidentally create multiple instances of this class as the compiler will check this.

The MergeSort works with Merge to merge sort the list of highscores. The LoadHighScores and SaveHighScores read and write to a text file respectively. The UpdateHighScores method will be run upon the application quitting (user dying). LoadHighScores() will be run upon starting the game. The high scores will be viewable by the NPC in the tutorial room when you collide with him.

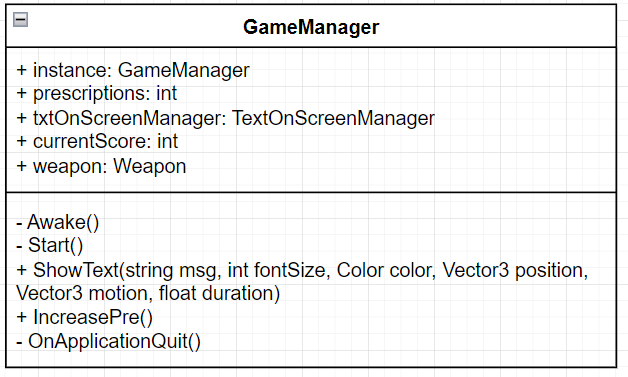
Pause



This class will manage the pausing of the game. I decided not to make this class static because of the fact that it has a more complicated checking system than the HighScoresManager. The GameManager will nonetheless be managing this script as it will have a field for it. This makes sense since I have more than 1 scene in my game and the pause script needs to be accessible in any scene, and my GameManager is set to DontDestroyOnLoad which means it and all components/scripts attached to it will not disappear upon entering a new scene.

The pausing of the game is actually in itself a very simple mechanism, simply changing the value of Time.timeScale to 0 will pause all movement inside the game. Setting it back to 1 will play the game and allow all movement to continue and occur. Input will still be allowed during the moment when Time.timeScale = 0 and therefore the user will still be able to unpause the game. Pausing and unpausing should happen through the escape key.

GameManager



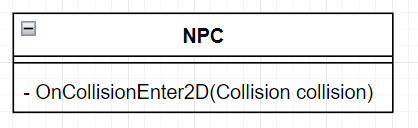
GameManager plays a crucial role in the game, it keeps track of the score of the player, the number of prescriptions they have, the high score, and controls the displaying of text on the screen.

The GameManager script follows the singleton design pattern, which means that there will only be one instance of it and that this instance can be accessed globally. It will have (as seen in the diagram) a public static reference to its own instance which will allow other scripts to access its public variables and methods without having to create a new instance of it. Awake() will ensure only one instance of GameManager is created by setting the instance variable to the GameManager’s script itself. DontDestroyOnLoad(gameObject) will also be located in Awake() and will prevent the GameManager from being destroyed across scenes; therefore guaranteeing that the GameManager remains persistent throughout the game.

OnApplicationQuit() runs right upon quitting the game and inside it will be the UpdateHighScores method.

Start() will have the LoadHighScores method on the other hand, and the list of these high scores (through HighScoresManager) will be displayed by the NPC upon collision with them.

NPC



When the player collides with the NPC the HighScoresManger list called highScores will be shown. As the message shown must be a string, the list will be converted to an array first, before it is joined using the string.Join which concatenates the elements of the array and splits them depending on the presence of a comma. To show the text, the ShowText method of GameManager will be called.

Chest Tutorial

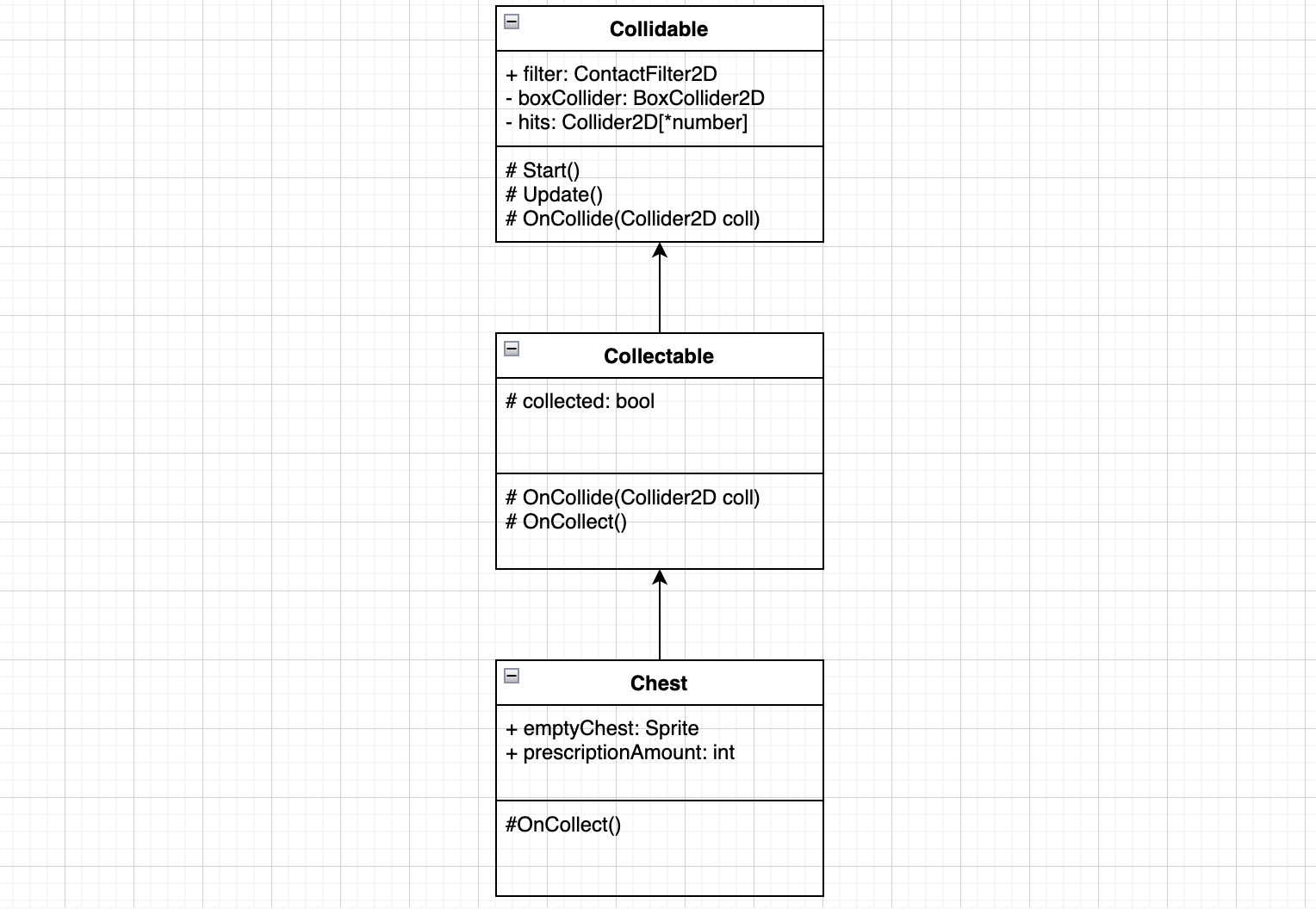
*Though this class inherits from Chest it is my own original idea.*

**

This is a very simple class that only slightly differs from the chest class with the OnCollect() method which is overridden, otherwise it inherits the attributes of the Chest class . This class will be used for a chest that I will add in the tutorial/spawn area of the game. It will grant the user a random number of prescriptions between around 1-30 and will output a message telling the user that they should attempt to collect as many of these as they can.

Chest - Some R

*The non-referenced part is further down in italics.*

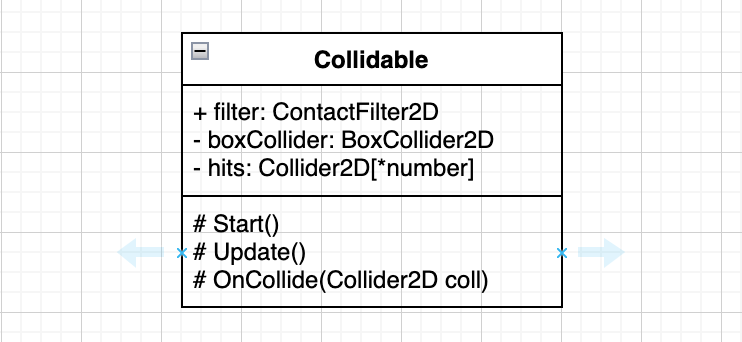


This class diagram is a final phase of what any collectable would look like. The Chest class inherits from the Collectable class, which inherits from the Collidable class - a prime example of what one of the main classes in the game will look like and how inheritance should be used throughout a game. The Chest has properties that mean it cannot be run through/phased through, and that means that when it collides with the player the player can receive rewards or currency based specifically on what they collided with. The OnCollect() method is overrode in the Chest class as collected is no longer set to true only. The code (which will be shown later) in summary checks if collected is true - which it is not at the time - and if not then it generates an amount of currency and this can be seen in the console as there is a Debug.Log within the if statement. After it checks whether collected is true or not, it of course assigns collected as true before doing this so that no repetitions or errors occur. The Sprite emptyChest is also run at this moment as the chest does not disappear in my game but has an empty sprite where there is nothing inside the chest, indicating its empty contents.

*This part is my own code/ideas - not copied/referenced.*

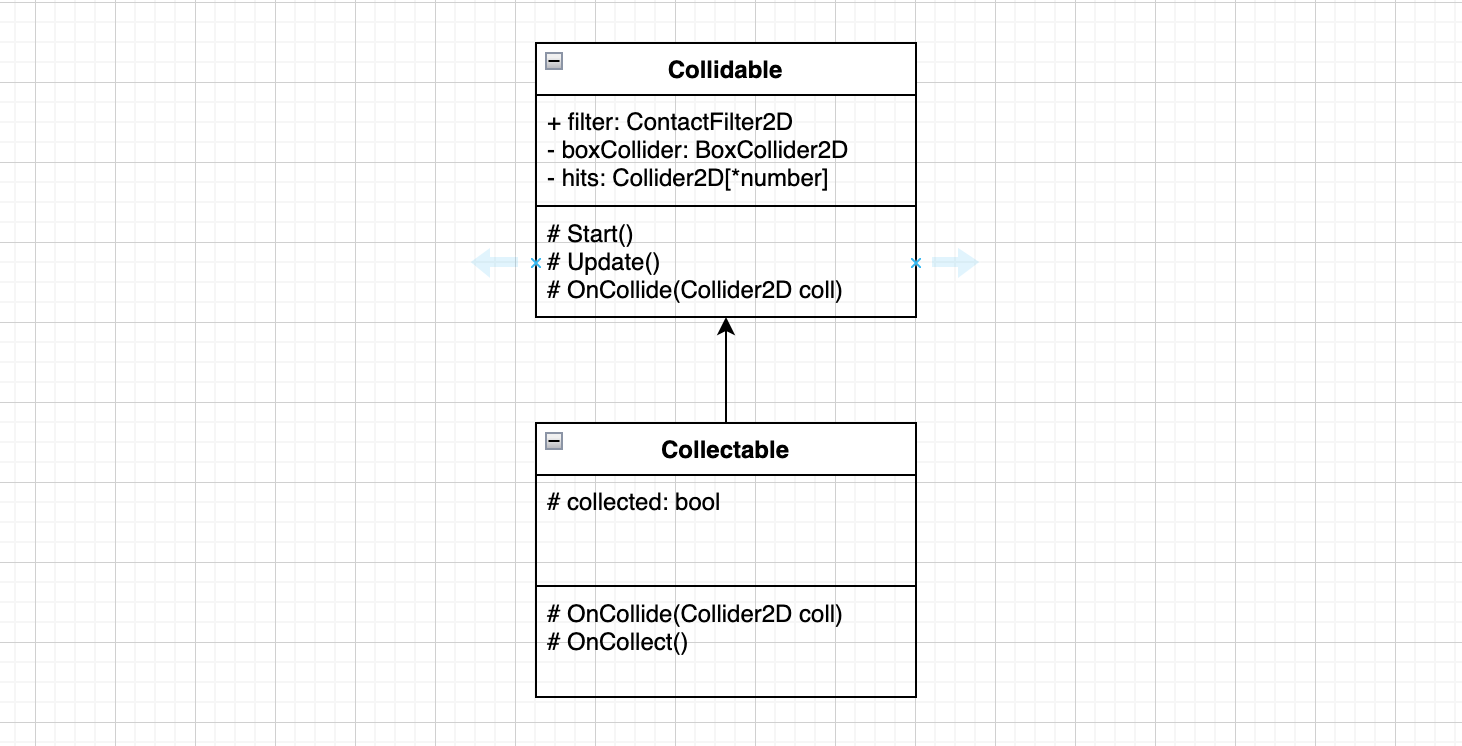
The prescriptionAmount field specifies how many prescriptions to give the player. Many ideas went into how to do this, but I arrived at a single conclusion, to make it public but random and not random at the same time. Depending on the value inputted in the public field (which of course can be changed from within Unity) the amount of prescriptions provided to the player is random or not random. A value less than 1 (I use 0) means that the amount of prescriptions generated will be a random number between 1 and 3, while any other number will mean that that number specifically will be the amount they will be given from that specific chest. I did this so that the player can be rewarded accordingly depending on the difficulty of acquiring certain chests in the game where it wouldn’t be motivating to the player if they only received little payment for their efforts.

Collision - R



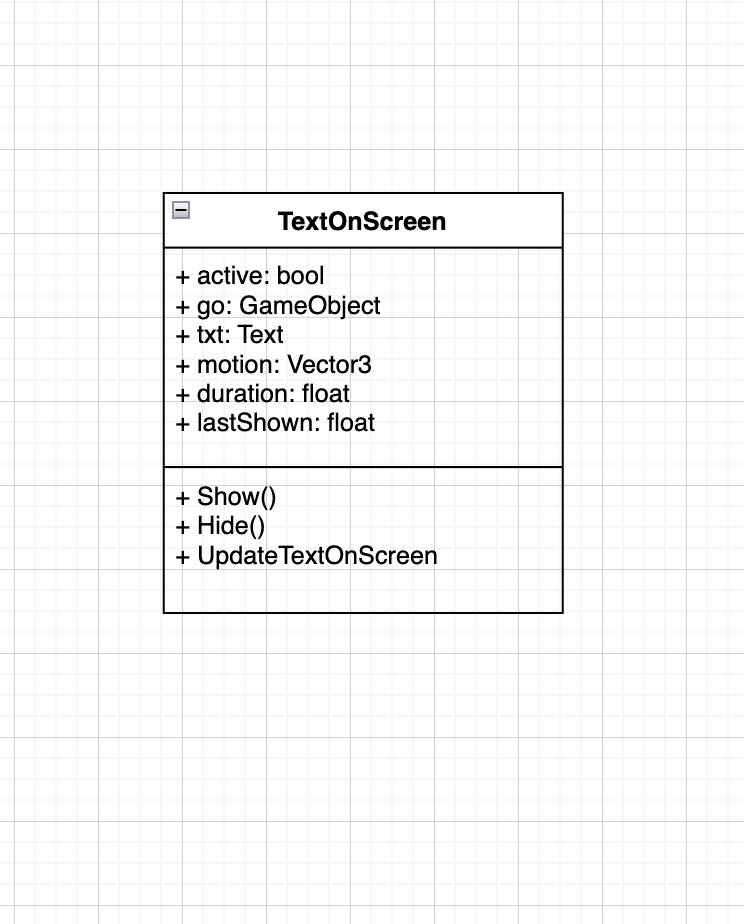
This is the first draft for the Collidable class. This class exists to let collision take place and certain actions, animations, events etc. to occur when collision between 2 things occur. It also stops the sprite/player from moving through other sprites. Through inheritance and the use of subclasses, this class will allow the sprite to interact with other sprites such as chests, healing fountains, mana fountains, whatever I want to program. This will give me lots of freedom and make it easier for me to code and specify what occurs when collision/interaction between 2 specific sprites occur. Using subclasses and inheritance and overriding later on will also make the coding easier for me, more readable, and will save time instead of writing a statement one after another. \*number within the array hits is the number of hits the program will be checking through at one time, or the number of collisions the program will be checking through at one time. This number shouldn’t be small or large as it being (too) small will limit the actual gameplay of the game as some interactions’ effects or collisions’ effects may not be rendered to the user, and it being (too) large will take up more memory which isn’t required for a simple game such as this. A more complex game with many types of interactions and collisions taking place at one time would require a larger number so that these can all be essentially calculated and rendered to the user, but my game is small and won’t have too many of these at once. Start and Update are used just so that the BoxCollider2D component can be set to boxCollider before anything else happens and so collisions are checked upon every frame, as checking for example 10 collisions every frame is more than enough for a game of my size and complexity.

Collecting - R



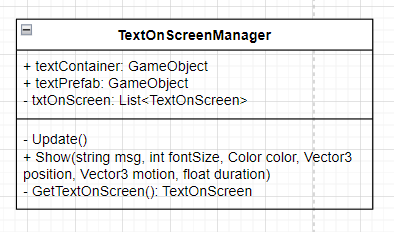
This class diagram represents the relationship between the Collidable class and the Collectable class. The Collectable class is clearly a subclass of the collidable class, almost the same but with a few small differences. The class has an override of the OnCollide(Collider2D coll) method whereby if the name of the thing colliding with the object is Player then the OnCollect() method is called and the value for collected - the boolean variable - is set to true. The use of this will be seen later on when collectables are added to the game, such as chests that contain some sort of game currency in them. The Collectable class also does not call the normal OnCollide(Collider2D coll) method and therefore will not execute the Debug.Log(coll.name) within the method.

Text On Screen - R



This class is the class for the on screen text to appear in-game due to certain collisions/interactions the player has. The fields used in this class can be easily explained through the methods that use them. Within the Show() method, active is set to true, lastShown is set to Time.time - the current time - and go.Setactive is set to the value of active, therefore is set to true. Within the Hide() method however, active is set to false and go.SetActive is to false as well as it again takes on the value of active. UpdateTextOnScreen() is the main method used in this, whereby it checks at the start whether active is false or not, if it is false, then it simply returns and does nothing. However, if it is true, then it checks the difference between the current time (Time.time) and the time the on-screen text was last shown, lastShown. It subtracts lastShown from Time.time and checks whether this is greater than the duration set by me, and if it is then the Hide() method is called and the text is hidden from the user. Simply put, I set a duration in which the on-screen text can last before disappearing again from the user's sight using the Show(), Hide(), and UpdateOnScreen() methods together.

Text On Screen Manager - R



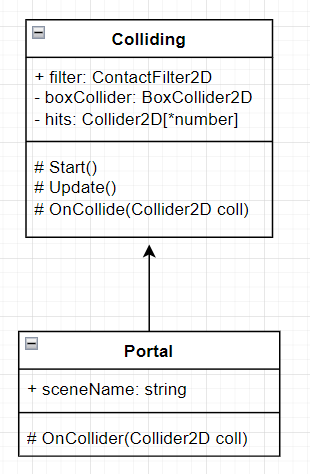
This class manages the display of the text on the screen in the game. It contains a public method Show() that takes in the following as arguments:

* String message
* Int fontSize
* Color color
* Vector3 position
* Vector3 motion
* Float Duration

When the method is called it creates a new text object, sets its properties based on the arguments, and displays it on the screen. The script uses a list of TextOnScreen objects to keep track of the text objects currently on the screen.

The script has a private method GetTextOnScreen() that gets an available TextOnScreen object from the list or creates a new one if none are available. The private Update() method calls the UpdateTextOnScreen() method from TextOnScreen for each text object currently on the screen.

Teleporter/Portal - Some R

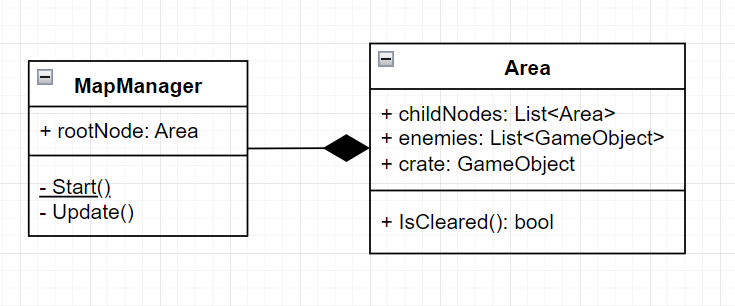


The portal works very simply by making use of a method in the SceneManager class of Unity. This is an in-built class that I will use to load the next scene for the next level of the game/part of the game. Simply put, when an object collides with the portal, if its name is Player then the player is sent to the next scene. I will use DontDestroyOnLoad on my player to make sure they keep their variables and statistics the same when they load into the next scene. The portal class inherits from the Colliding class as seen above as well.

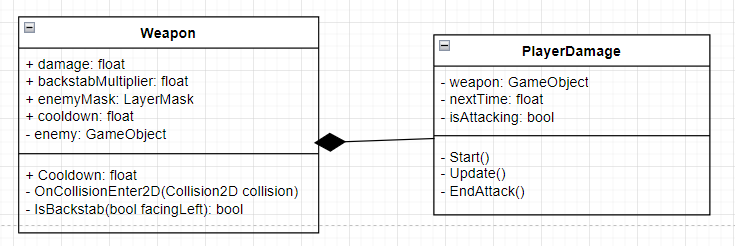
## UML Class Diagrams - Relationships

Map

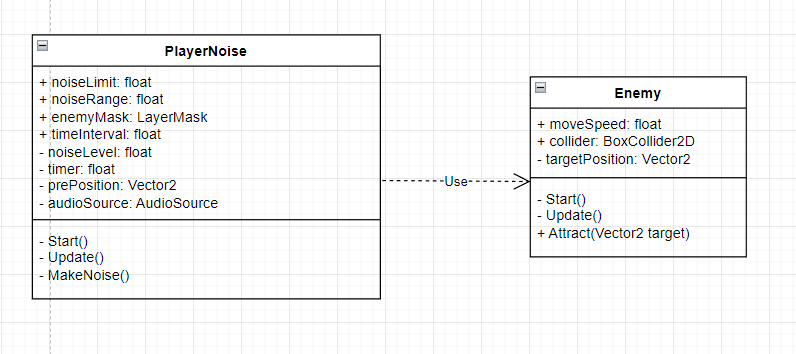
*This might look wrong at first glance, but remember that in my game the Area class is designed to be used as part of the larger map that is represented by the MapManager class.*



PlayerDamage and Weapon



PlayerNoise and Enemy



# Technical Solution

*The order of the following scripts/classes is the same as that of the UML Class Diagrams - Individual. The scripts also follow the same rule as the Documented Solution, if there is R after a dash in the title of the script then it is a script that has a reference attached to it which can be found in the References section of this paper.*

## Player.cs - R

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class Player : MonoBehaviour {   private BoxCollider2D boxCollider;  private Vector2 movePlayer;  private RaycastHit2D hit;  private float moveSpeed;    private void Start()  {  //DontDestroyOnLoad(gameObject); hasn't been used as  //this would cause the player to appear in a random location in the next scene  //when they teleport  boxCollider = GetComponent<BoxCollider2D>();  }   //private RaycastHit2D GetHit()  //{  //return hit;  //}   /\*private void FixedUpdate()  {  float x = Input.GetAxisRaw("Horizontal");  float y = Input.GetAxisRaw("Vertical");    movePlayer = new Vector2(x, y);  Move();  }\*/    private void FixedUpdate()  {  if (hit.collider == null)  {  //Make the sprite move  transform.Translate(movePlayer.x, movePlayer.y \* Time.deltaTime, 0);   }   float x = Input.GetAxisRaw("Horizontal");  float y = Input.GetAxisRaw("Vertical");  //Resetting movePlayer  movePlayer = new Vector2(x, y);   //Swapping sprite direction  if (movePlayer.x > 0)  transform.localScale = Vector2.one; //If d is pressed then sprite faces right  else if (movePlayer.x < 0)  transform.localScale = new Vector2(-1, 1); //If a is pressed then sprite faces left  hit = Physics2D.BoxCast(transform.position, boxCollider.size, 0, new Vector2(0, movePlayer.y), Mathf.Abs(movePlayer.y \* Time.deltaTime), LayerMask.GetMask("Actor", "Blocking"));  Move(hit);  //if (hit.collider == null)  {  //Make the sprite move  //transform.Translate(0, movePlayer.y \* Time.deltaTime, 0);   }  //Same thing but for the x-axis  hit = Physics2D.BoxCast(transform.position, boxCollider.size, 0, new Vector2(movePlayer.x, 0), Mathf.Abs(movePlayer.x \* Time.deltaTime), LayerMask.GetMask("Actor", "Blocking"));  Move(hit);  //if (hit.collider == null)  {  //Make the sprite move  //transform.Translate(movePlayer.x \* Time.deltaTime, 0, 0);   }  }  private void Move(RaycastHit2D hit)  {  if (hit.collider == null)  {  transform.Translate(movePlayer.x \* Time.deltaTime, movePlayer.y \* Time.deltaTime, 0);  }  } } |
| --- |

## PlayerMovement.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine; using UnityEngine.UIElements;  public class PlayerMovement : MonoBehaviour {  public float speed = 1f; // Speed at which the player moves  public LayerMask collisionMask; // Layers player can collide with   private Vector2 moveDirection = Vector2.zero; // The direction the player is moving  private Rigidbody2D rb2D; // The Rigidbody2D component attached to the player object   void Start()  {  rb2D = GetComponent<Rigidbody2D>(); // Get the Rigidbody2D component from the player object  DontDestroyOnLoad(gameObject); // Do not destroy this object when a new scene is loaded  }   void Update()  {  // Get input from user  moveDirection.x = Input.GetAxisRaw("Horizontal"); // Get the horizontal input from the user  moveDirection.y = Input.GetAxisRaw("Vertical"); // Get the vertical input from the user  moveDirection = new Vector2(moveDirection.x, moveDirection.y).normalized; // Normalize the direction vector   // Swap sprite direction based on input  if (moveDirection.x <= 0)  {  if (moveDirection.x < 0)  transform.localScale = new Vector2(-1, 1); // If 'a' is pressed then sprite faces left  }  else  {  transform.localScale = Vector2.one; // If 'd' is pressed then sprite faces right  }  moveDirection \*= speed; // Multiply the direction vector by the speed value to get the movement vector   // Check for collisions against collisionMask using BoxCast  RaycastHit2D hit = Physics2D.BoxCast(transform.position, GetComponent<BoxCollider2D>().size, 0, moveDirection, Mathf.Abs(speed \* Time.deltaTime), collisionMask); // Cast a box in the direction of movement to check for collisions   if (hit.collider != null)  {  moveDirection = Vector2.zero; // Stop the player's movement if there is a collision  }   rb2D.velocity = moveDirection; // Set the player's velocity to the movement vector  } } |
| --- |

## PlayerDamage.cs

| using System.Collections; using System.Collections.Generic; using System.Runtime.CompilerServices; using UnityEngine;  // Controls when the player can do damage public class PlayerDamage : MonoBehaviour {  private GameObject weapon; // Reference to the player's weapon  private float nextTime = 0; // The time until the player can attack again  private bool isAttacking = false;   private void Start()  {  weapon = GameObject.Find("Weapon");  }   private void Update()  {  if (Time.time >= nextTime) // Check if enough time has passed since the last attack  {  if (Input.GetButtonDown("Fire1") && !isAttacking) // Check if the attack button has been pressed and the player is not already attacking  {  isAttacking = true;  nextTime = Time.time + weapon.GetComponent<Weapon>().Cooldown; // Set the next attack time  Invoke("EndAttack", 0.2f); // Call EndAttack after 0.2 seconds  weapon.GetComponent<BoxCollider2D>().enabled = true; // Enable the weapon's collider to detect collisions  }  }  }   // Disable the weapon's collider and set attacking to false  private void EndAttack()  {  weapon.GetComponent<BoxCollider2D>().enabled = false;  isAttacking = false;  } } |
| --- |

## Weapon.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class Weapon : MonoBehaviour {  public float damage = 1; // Base damage dealt  public float backstabMultiplier = 3; // The multiplier applied to damage if the attack is a backstab  public LayerMask enemyMask; // The layer mask used to detect enemies  public float cooldown = 3f; // Cooldown till next attack   private GameObject enemy; // The enemy that the player is currently attacking   // Getter for cooldown value  public float Cooldown  {  get { return cooldown; }  }   // This method runs when a collision occurs  private void OnCollisionEnter2D(Collision2D collision)  {  if (enemyMask == (enemyMask | (1 << collision.gameObject.layer))) // Check if the collision is with an enemy layer  {  enemy = collision.gameObject; // Set the GameObject the player collided with to enemy   // Check which way the enemy is facing  bool facingLeft = enemy.transform.localScale.x < 0;   // Calculate damage dealt based on whether the weapon is behind the enemy  if (IsBackstab(facingLeft))  {  damage \*= backstabMultiplier++; // If the attack is a backstab then multiply the damage by the backstab multiplier and increment by 1  }   // Deal damage to the enemy  Health enemyHealth = enemy.GetComponent<Health>(); // Get the enemy's Health component  if (enemyHealth != null) // Check if the enemy has a Health component  {  enemyHealth.TakeDamage(damage); // Deal damage to the enemy  }  damage = 1;  }  }   // Check if the attack is a backstab based on the weapon's direction and the enemy's facing direction  private bool IsBackstab(bool facingLeft)  {  // Direction of the player's weapon  Vector2 weaponDirection = transform.right;   // Direction of weapon to enemy  Vector2 weaponToEnemy = transform.position - enemy.transform.position;   // Flip the weaponToEnemy vector if the enemy is facing left  if (facingLeft)  {  weaponToEnemy = -weaponToEnemy;  }   // Calculate dot product  float dotProduct = Vector2.Dot(weaponDirection, weaponToEnemy.normalized);   // Check if the weapon is behind the enemy  return dotProduct < 0;  } } |
| --- |

## PlayerNoise.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine; using UnityEngine.AI;  public class PlayerNoise : MonoBehaviour {  public float noiseLimit; // The maximum amount of noise the player can make before the enemies are alerted  public float noiseRange; // The range at which enemies can detect the player's noise  public LayerMask enemyMask; // Layer mask for enemies  public float timeInterval = 0.5f; // Time interval at which the noise level is calculated   private float noiseLevel; // The current level of noise made by the player  private float timer; // The timer used to calculate the noise level  private Vector2 prePosition; // The previous position of the player  private AudioSource audioSource; // The audio source component attached to the player   private void Start()  {  timer = 0.0f;  prePosition = transform.position;  audioSource = GetComponent<AudioSource>();  }   private void Update()  {  timer += Time.deltaTime; // Set timer to the current time + timer   if (timer >= timeInterval) // If the timer (timeElapsed) is greater than the specified interval  {  // Calculate noise level based on distance covered every time interval  float distanceCovered = Vector2.Distance(transform.position, prePosition);  noiseLevel += Mathf.Min(distanceCovered, noiseLimit);  timer = 0.0f;  prePosition = transform.position;   if (noiseLevel > noiseLimit) // If the amount of noise being made is greater than the limit  {  // Emit sound where the volume of said sound is based on how much noise the user is making  float volume = (noiseLevel - (noiseLimit \* 1.2f)) / noiseLimit;  audioSource.volume = Mathf.Clamp(volume, 0, 1);  noiseRange = Mathf.Clamp(volume, 0, 1);  if (!audioSource.isPlaying)  {  audioSource.Play();  }  MakeNoise();   // Reset noise level  noiseLevel = 0.0f;  }  }  }   // Attracts nearby enemies towards the player based on the noise range  private void MakeNoise()  {  // Find all colliders in the radius noiseRange with layer Enemy  Collider2D[] enemies = Physics2D.OverlapCircleAll(transform.position, noiseRange, enemyMask);   // Attract each enemy to the position of the player   foreach (Collider2D enemy in enemies)  {  Enemy enemyScript = enemy.GetComponent<Enemy>();  enemyScript.Attract(transform.position);  }  } } |
| --- |

## Enemy.cs

| using System.Collections; using System.Collections.Generic; using System.Drawing; using UnityEngine;  public class Enemy : MonoBehaviour {  public float moveSpeed; // The speed at which the enemy moves  public new BoxCollider2D collider; // The collider attached to the enemy   private Vector2 targetPosition; // The position the enemy is moving towards   private void Start()  {  targetPosition = transform.position; // Initialize the target position to the enemy's current position  collider = gameObject.GetComponent<BoxCollider2D>(); // Get the collider component on the enemy game object  }   // Checking for collisions  private void Update()  {  // Check if there's a wall in the way  Vector2 direction = (targetPosition - (Vector2)transform.position).normalized; // The direction the enemy is moving  Vector2 size = collider.size; // The size of the enemy's collider  float distance = Vector2.Distance(transform.position, targetPosition); // The distance between the enemy and the target position  RaycastHit2D hit = Physics2D.BoxCast(transform.position, size, 0, direction, distance, LayerMask.GetMask("Blocking")); // Cast a box to check for obstacles in the enemy's path   // If there's no wall, move towards the target position  if (hit.collider == null)  {  transform.position = Vector2.MoveTowards(transform.position, targetPosition, moveSpeed \* Time.deltaTime);  }  // If there's a wall, stop moving  else  {  return;  }  }   public void Attract(Vector2 target)  {  // Update the target position to move towards the player's position  targetPosition = target;  } } |
| --- |

## EnemyDamage.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class EnemyDamage : MonoBehaviour {  public float damage = 1;   private void OnCollisionEnter2D(Collision2D collision)  {  // If this object collides with the Player object  if (collision.gameObject.name == "Player")  {  // Get the Health component of the Player object  Health playerHealth = collision.gameObject.GetComponent<Health>();  // Call the TakeDamage function of the Health component to damage the player  playerHealth.TakeDamage(damage);  }  } } |
| --- |

## Health.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class Health : MonoBehaviour {  // The maximum/starting amount of health this entity can have/has  public float startHealth = 10f;   // The current amount of health this entity has  public float currentHealth;   // A reference to the Area script  public Area area;   private void Start()  {  // Set the current health to the maximum health at the start of the game  currentHealth = startHealth;  }   // Method for taking damage  public void TakeDamage(float damage)  {  // Subtract the damage from the current health  currentHealth -= damage;   // If the current health is less than or equal to 0, call the Die method  if (currentHealth <= 0)  {  Die();  }  }   // Method for handling death  private void Die()  {  // If the Player dies then destroy the Player gameObject and end the game/quit the application  if (gameObject.name == "Player")  {  Destroy(gameObject);  Application.Quit();  UnityEditor.EditorApplication.isPlaying = false;  }   // Search for this enemy in the list of enemies managed by the Area script, and remove it from the list  for (int i = 0; i < area.enemies.Count; i++)  {  if (area.enemies[i].name == gameObject.name)  {  area.enemies.RemoveAt(i);  break;  }  }   Destroy(gameObject);  }  } |
| --- |

## CameraMotion.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class CameraMotion : MonoBehaviour {  /\*public Transform lookAt;  //This is the amount of space on the axes the character can move before the camera starts moving  public float boundX = 0.3f;  public float boundY = 0.15f;\*/  public Transform lookAt;   private void Start()  {  // Find the position of the player (object) and set it as the target for the camera to follow  lookAt = GameObject.Find("Player").transform;  }   /\*private void LateUpdate()  {  Vector3 delta = Vector3.zero;   //This is to check if we are inside the bound on the X-axis  //transform.position is the centre/midpoint of where the camera is  //lookAt.position is the where the character is (in their respective axes)  float deltaX = lookAt.position.x - transform.position.x;  if (deltaX > boundX || deltaX < -boundX)  {  if (transform.position.x < lookAt.position.x)  {  delta.x = deltaX - boundX;  }  else  {  delta.x = deltaX + boundX;  }  }   //This is to check if we are inside the bound on the Y-axis  float deltaY = lookAt.position.y - transform.position.y;  if (deltaY > boundY || deltaY < -boundY)  {  if (transform.position.y < lookAt.position.y)  {  delta.y = deltaY - boundY;  }  else  {  delta.y = deltaY + boundY;  }  }   transform.position += new Vector3(delta.x, delta.y, 0);  }\*/    private void LateUpdate()  {  // Update the position of the camera to be the same as the position of the player  transform.position = new Vector3(lookAt.position.x, lookAt.position.y, -10);  } } |
| --- |

## Area.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class Area : MonoBehaviour {  public List<Area> childNodes; // A list of child nodes of this area.  public List<GameObject> enemies; // A list of enemies in this area.  public GameObject crate; // A crate object in this area.  //public bool cleared;   // Constructor for the Area class that sets initial values for its fields.  // Only used when MapManager is privately creating the Areas  // Otherwise it is redundant  /\*public Area(List<GameObject> enemies, GameObject crates)  {  this.childNodes = new List<Area>();  this.enemies = enemies;  this.crate = crates;  }   // Adds a child node to this area.  // This is also redundant if areas are created through Unity inspector  public void AddNode(Area node)  {  childNodes.Add(node);  }\*/   // Returns true if all enemies in this area and its child nodes have been defeated, and false otherwise.  public bool IsCleared()  {  if (enemies.Count > 0) // If there are still enemies in this area, it is not yet cleared and return false.  {  return false;  }   foreach (Area node in childNodes) // Check if all child nodes are cleared.  {  if (!node.IsCleared()) // If any child node is not cleared, this area is not cleared.  {  return false;  }  }   // Disable the collider of the crate.  Collider collider = crate.GetComponent<Collider>();  if (collider != null)  {  collider.enabled = false;  }   // Hide the crate object.  crate.SetActive(false);   return true; // All enemies in this area and its child nodes have been defeated, so this area is cleared.  }  } |
| --- |

## MapManager.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine; using static UnityEditor.VersionControl.Asset;  public class MapManager : MonoBehaviour {  public Area rootNode; // Instance of Area class that represents rood node of map   /\*private void Start()  {  // Creating the root node  rootNode = new Area(null, null);   // Creating the child nodes  Area area1 = new Area(null, null);  Area area1\_1= new Area(null, null);   // This is what the code would look like for creation of an area  Area area1\_2= new Area(null, GameObject.Find("crates1"));    Area area1\_3= new Area(null, null);   Area area2= new Area(null, null);  Area area2\_1= new Area(null, null);  Area area2\_2 = new Area(null, null);   Area area3= new Area(null, null);  Area area3\_1 = new Area(null, null);  Area area3\_2= new Area(null, null);   Area area4= new Area(null, null);  Area area4\_1 = new Area(null, null);  Area area4\_2= new Area(null, null);   Area area5= new Area(null, null);   // Adding the child nodes to the root node  rootNode.AddNode(area1);  rootNode.AddNode(area2);  rootNode.AddNode(area3);  rootNode.AddNode(area4);  rootNode.AddNode(area5);   // Adding child nodes to parent nodes  area1.AddNode(area1\_1);  area1.AddNode(area1\_2);  area1.AddNode(area1\_3);   area2.AddNode(area2\_1);  area2.AddNode(area2\_2);   area3.AddNode(area3\_1);  area3.AddNode(area3\_2);   area4.AddNode(area4\_1);  area4.AddNode(area4\_2);   GameObject crateToDestroy = rootNode.childNodes[0].childNodes[1].crate;  Debug.Log(crateToDestroy);  GameObject.Destroy(crateToDestroy);  Debug.Log("Its Working");    }\*/   private void Update()  {  // Regularly check if map has been cleared  // If so, create a message that says they have been cleared  if (rootNode.IsCleared())  {  Debug.Log("They have all been cleared!");  }   } } |
| --- |

## HighScoresManager.cs

| using System.Collections.Generic; using System.IO;  public static class HighScoresManager {  public static List<int> highScores = new List<int>(); // List to hold the high scores   // Merge sort algorithm implementation  public static List<int> MergeSort(List<int> originalList)  {  if (originalList.Count <= 1)  {  return originalList;  }   int middleOfList = originalList.Count / 2;  List<int> leftSide = originalList.GetRange(0, middleOfList);  List<int> rightSide = originalList.GetRange(middleOfList, originalList.Count - middleOfList);   leftSide = MergeSort(leftSide);  rightSide = MergeSort(rightSide);   return Merge(leftSide, rightSide);  }   // Helper function to merge two sorted lists into one sorted list  private static List<int> Merge(List<int> left, List<int> right)  {  List<int> sortedList = new List<int>();  int leftIndex = 0, rightIndex = 0;   while (leftIndex < left.Count && rightIndex < right.Count)  {  sortedList.Add(left[leftIndex] < right[rightIndex] ? left[leftIndex++] : right[rightIndex++]);  }   sortedList.AddRange(left.GetRange(leftIndex, left.Count - leftIndex));  sortedList.AddRange(right.GetRange(rightIndex, right.Count - rightIndex));   return sortedList;  }   // Load the high scores from the text file  public static void LoadHighScores()  {  string path = "highscores.txt"; // Path to the text file holding the high scores   // Create the file if it doesn't exist  if (!File.Exists(path))  {  using (FileStream fs = File.Create(path))  {  fs.Close();  }  }   // Read the scores from the text file and add them to the list of high scores  using (StreamReader reader = new StreamReader(path))  {  while (!reader.EndOfStream)  {  string line = reader.ReadLine();  int highScore = int.Parse(line);  highScores.Add(highScore);  }  }  }   // Save the high scores to the text file  public static void SaveHighScores(List<int> highScores)  {  string path = "highscores.txt"; // Path to the text file holding the high scores   // Write the high scores to the text file  using (StreamWriter writer = new StreamWriter(path))  {  foreach (int highScore in highScores)  {  writer.WriteLine(highScore);  }  }  }   // Update the high scores list with a new score, sort it, and trim it to keep only the top 5 scores  public static void UpdateHighScores(int newHighScore)  {  highScores = new List<int>(); // Clear the existing high scores list  LoadHighScores(); // Load the high scores from the text file  highScores.Add(newHighScore); // Add the new high score to the list  highScores = MergeSort(highScores); // Sort the list using the merge sort algorithm  if (highScores.Count > 5) // Trim the list to keep only the top 5 scores  {  highScores.RemoveAt(0);  }  SaveHighScores(highScores); // Save the updated high scores to the text file  } } |
| --- |

## isPause.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class Pause : MonoBehaviour {  // Tracks whether game is paused or not  public bool isPaused;   private void Update()  {  // If the Escape key is pressed  if (Input.GetKeyDown(KeyCode.Escape))  {  // If the game is already paused  if (isPaused)  {  // Resume the game  ResumeGame();  }  // If the game is not paused  else  {  // Pause the game  PauseGame();  }  }  }   public void PauseGame()  {  // Set the game's time scale to 0 to pause all movement  Time.timeScale = 0.0f;   // Update the isPaused variable to reflect that the game is now paused  isPaused = true;  }   public void ResumeGame()  {  // Set the game's time scale back to 1 to resume movement  Time.timeScale = 1.0f;   // Update the isPaused variable to reflect that the game is no longer paused  isPaused = false;  } } |
| --- |

## GameManager.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine; using UnityEngine.SceneManagement;  public class GameManager : MonoBehaviour {  // A static reference to the GameManager instance so it can be accessed from other scripts  public static GameManager instance;   // The number of prescriptions the player has collected  public int prescriptions;   // A reference to the TextOnScreenManager script for displaying text on screen  public TextOnScreenManager txtOnScreenManager;   // The player's current score  public int currentScore;   // A reference to the PlayerDamage script for calculating the score  public Weapon weapon;   // Awake is called when the script instance is being loaded  private void Awake()  {  // Set the instance variable to this script so it can be accessed from other scripts  instance = this;   // Don't destroy the GameManager when a new scene is loaded, so it stays persistent throughout the game  DontDestroyOnLoad(gameObject);  }   private void Start()  {  // Load the high scores from a text file as soon as the game starts  HighScoresManager.LoadHighScores();  }   // A public method for displaying text on screen  public void ShowText(string msg, int fontSize, Color color, Vector3 position, Vector3 motion, float duration)  {  // Call the Show method of the TextOnScreenManager script  txtOnScreenManager.Show(msg, fontSize, color, position, motion, duration);  }   // A method for increasing the number of prescriptions the player has collected and updating the score  public void IncreasePre(int amount)  {  // Increment the number of prescriptions  prescriptions += amount;   // Calculate the score based on the number of prescriptions collected and the backstab multiplier from the PlayerDamage script  float score = amount \* weapon.backstabMultiplier;   // Round the score to the nearest integer and set it as the current score  currentScore = Mathf.RoundToInt(score);  }   // Called when the application is about to quit  private void OnApplicationQuit()  {  // Update and save the high scores as soon as the game closes  HighScoresManager.UpdateHighScores(currentScore);  }  } |
| --- |

## NPC.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class NPC : MonoBehaviour {  // This method is called when the NPC collides with another object  private void OnCollisionEnter2D(Collision2D collision)  {  string highScoresText = string.Empty; // Create a new empty string  // Format string as the highScores list vertically  foreach (int score in HighScoresManager.highScores)  {  highScoresText += $"{score.ToString()}\n"; // Separate each score by a newline  }   // Access the static GameManager instance and call the ShowText method on the string string  GameManager.instance.ShowText(highScoresText, 20, Color.white, transform.position, Vector3.zero, 1.5f);  } } |
| --- |

## ChestTutorial.cs

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class ChestTutorial : Chest {  // Called when the player collects the Tutorial Chest  protected override void OnCollect()  {  if (!collected)  {  collected = true;  prescriptionAmount = Random.Range(1, 25);  GetComponent<SpriteRenderer>().sprite = emptyChest;  GameManager.instance.ShowText($"Grant {prescriptionAmount} pr...", 13, Color.yellow, transform.position, Vector3.up \* 25, 1.5f);  GameManager.instance.ShowText("These will increae your score, so make sure to collect as many as you can!", 20, Color.white, transform.position, Vector3.down \* 10, 4f);  }  } } |
| --- |

## Chest.cs - Some R

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class Chest : Collecting {  // Sprite to show when the chest is empty  public Sprite emptyChest;  // Amount of prescription the player gains when they collect the chest  public int prescriptionAmount;   // Called when the player collects the chest  protected override void OnCollect()  {  // If the chest hasn't already been collected  if (!collected)  {  collected = true;  // If the chest hasn't already been assigned a prescription amount  if (prescriptionAmount < 1)  {  // Assign a random number between 1 and 4 to prescriptionAmount  prescriptionAmount = Random.Range(1, 4);  }  // Change the sprite to the emptyChest sprite  GetComponent<SpriteRenderer>().sprite = emptyChest;  // Show a text message indicating the player has gained prescriptionAmount prescriptions  GameManager.instance.ShowText($"Gained {prescriptionAmount} pr...", 24, Color.yellow, transform.position, Vector3.up \* 25, 1.5f);  // Increase the player's prescription count  GameManager.instance.IncreasePre(prescriptionAmount);  }  } } Portal.cs - Some R  | using System.Collections; using System.Collections.Generic; using UnityEngine; using UnityEngine.SceneManagement;  public class Portal : Colliding {  // This is a public variable named sceneName of type string, which will be used to store the name of the scene the player will be teleported to when they collide with the portal.  // It's marked as public so that it can be easily changed in the Unity editor.  public string sceneName;   // Called when the player collides with the portal  protected override void OnCollide(Collider2D coll)  {  // Check if the collider that the portal has collided with is named "Player"  if (coll.name == "Player")  {  // If the collider is the player, load the scene specified in the sceneName variable using the SceneManager class in Unity.  // This will change the current scene to the scene specified in sceneName.  SceneManager.LoadScene(sceneName);  }  } } | | --- |   *The following scripts are all referenced scripts - just like in the UML diagrams.* |
| --- | --- |

## Colliding.cs - R

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class Colliding : MonoBehaviour {  // The ContactFilter2D object that determines which colliders should trigger collisions  public ContactFilter2D filter;   // The BoxCollider2D component attached to the game object  private BoxCollider2D boxCollider;   // An array of Collider2D objects that stores information about the colliders that have collided with the box collider  private Collider2D[] hits = new Collider2D[10];   protected virtual void Start()  {  // Initialises boxCollider variable  boxCollider = GetComponent<BoxCollider2D>();  }   // This method is called every frame and checks for collisions between the box collider and other colliders  protected virtual void Update()  {  // Checking for collisions  boxCollider.OverlapCollider(filter, hits);   // Loop through each collider that collided with the box collider  for (int i = 0; i < hits.Length; i++)  {  // If the collider is null, skip to the next one  if (hits[i] == null)  {  continue;  }  else  {  // Call the OnCollide method and pass the collider as a parameter  OnCollide(hits[i]);   // Clean up the array for the next use  hits[i] = null;  }  }  }   // This method is called whenever the box collider collides with another collider  // It shows the name of the collider that it collided with  // This method can be overridden in subclasses to do certain things when colliding with certain objects or players  protected virtual void OnCollide(Collider2D coll)  {  Debug.Log(coll.name);  } } |
| --- |

## Collecting.cs - R

| using System.Collections; using System.Collections.Generic; using UnityEngine;  public class Collecting : Colliding {  // A boolean that keeps track of whether the object has been collected  protected bool collected;   // This method is called whenever the object collides with another collider  // If the collider is the player, the OnCollect method is called  protected override void OnCollide(Collider2D coll)  {  if (coll.name == "Player")  {  OnCollect();  }  }   // This method is called when the object has been collected  protected virtual void OnCollect()  {  collected = true;  } } |
| --- |

## TextOnScreen.cs - R

| using UnityEngine; using UnityEngine.UI;  public class TextOnScreen {  public bool active; // Determines whether text is being displayed or not  public GameObject go; // GameObject that represents the text object  public Text txt; // Text component of GameObject  public Vector3 motion; // Direction and speed that text will move on screen  public float duration; // Duration of text appearance  public float lastShown; // Time when text was last shown   // This method is used to show the text on the screen  public void Show()  {  active = true;  lastShown = Time.time;  go.SetActive(active);  }   // This method is used to hide the text on the screen  public void Hide()  {  active = false;  go.SetActive(active);  }    // This method is called to update the position of the text on the screen  public void UpdateTextOnScreen()  {  if (!active)  return;   // If duration has elapsed  if (Time.time - lastShown > duration)  Hide();   // Move the text on screen by motion \* time elapsed since last frame  go.transform.position += motion \* Time.deltaTime;  } } |
| --- |

## TextOnScreenManager.cs - R

| using System.Collections; using System.Collections.Generic; using UnityEngine; using UnityEngine.UI;  public class TextOnScreenManager : MonoBehaviour {  public GameObject textContainer;  public GameObject textPrefab;   private List<TextOnScreen> txtOnScreen = new List<TextOnScreen>();   private void Update()  {  // Call the UpdateTextOnScreen method for each text object currently on screen  foreach (TextOnScreen txt in txtOnScreen)  txt.UpdateTextOnScreen();  }   // Show a new text object on screen  public void Show(string msg, int fontSize, Color color, Vector3 position, Vector3 motion, float duration)  {  // Get an available TextOnScreen object or create a new one if none are available  TextOnScreen txtOnScreen = GetTextOnScreen();   // Set the text, font size, and color of the text object  txtOnScreen.txt.text = msg;  txtOnScreen.txt.fontSize = fontSize;  txtOnScreen.txt.color = color;   // Set the position, motion, and duration of the text object  txtOnScreen.go.transform.position = Camera.main.WorldToScreenPoint(position);  txtOnScreen.motion = motion;  txtOnScreen.duration = duration;   // Show the text object on screen  txtOnScreen.Show();  }   // Get an available TextOnScreen object or create a new one if none are available  private TextOnScreen GetTextOnScreen()  {  TextOnScreen txt = txtOnScreen.Find(t => !t.active);   if (txt == null)  {  txt = new TextOnScreen();  txt.go = Instantiate(textPrefab);  txt.go.transform.SetParent(textContainer.transform);  txt.txt = txt.go.GetComponent<Text>();   txtOnScreen.Add(txt);  }   return txt;  } } |
| --- |

# Testing

*If a test has a video, it will have V in brackets before the test description, and FV if the test is present in the final testing video. The timestamp for 2 min 20 seconds in will look like this 2:30, and will come right after the FV identifier. The videos related to this section are named by the title of the section, and then the test number. For example, the video for Test 1 for the sound and player movement system is named Sound and player movement system - Test 1.*

## Sound and player movement system

Test whether sound is indeed made when the character moves. Later on tests will check whether the character makes more sound when moving faster, and whether enemies are attracted to his position. The radius of the attraction of enemies should increase depending on the distance travelled as well.

(V)Test 1: it turns out that the character only makes sound when he collides with the wall or an object. This could be useful for when he receives damage as this could be an indicator of this, but this needs to be fixed nonetheless.

Test 2: through debugging, I found that there is a serious error as to when rcHit2d.collider = null. I might need to change this to boxCast instead, but will experiment by creating raycasts in each corner before this happens to see what occurs.

(V)Test 3: the update of the Player script to PlayerMovement fixed the problem I had. The character now makes an ominous sound when moving too much, and this sound attracts enemies to his position when he makes the sound. The radius of this sound is also dependent/proportional to the amount of noise the player makes.

## Collision mechanism

Check that the character collides properly with the walls and objects within the game, at a later stage this will also include collisions where the character receives damage - and this is almost done.

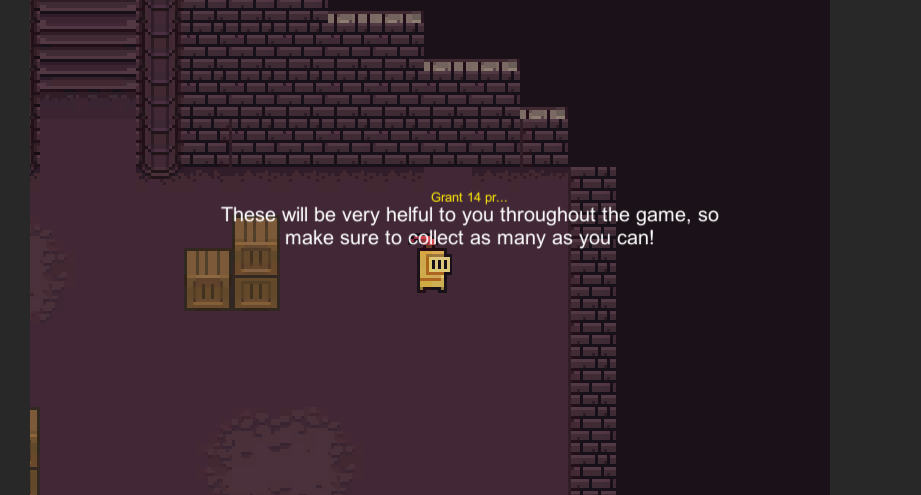
*This was when I was using raycast (rays) for collision detection as opposed to boxcast (boxes).*

(V)Test 1: as seen in the Sound and player movement system - Test 1 video, there is a problem. I have tested multiple methods of checking whether collision detection is working properly, but in each one the character always seems to phase halfway through the wall. I also checked whether maybe the centre of the character is where the ray is originating from, but cannot exactly tell through my tests. I believe that it is best to switch to using BoxCast for collision detection due to the problems occurring with RayCast. BoxCast is also more compatible with 2D games in general and is more widely used in these games, so it makes more sense to use it.

(V)Test 2: This was clearly very successful. The player collides properly with the walls now and does not phase through any of the objects. The change to BoxCast was the right call.

## Text

Make sure that text appears as it should and as programmed, such as in the subclasses where the text is different from the generic one. Also check whether the random generator works or not for chest prescriptions.



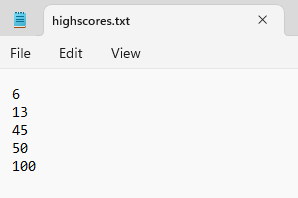
Test 1: As seen above, the text appears as it should. The random prescriptions are also working as seen below.

| Input | Expected | Output | Meaning |
| --- | --- | --- | --- |
| 12 | Random(1,25) | 3 | This works as it should |
| 0 | Random(1,25) | 14 | This works as it should |
| Chest class (above was the chest tutorial class, which works a little differently) | | | |
| 0 | Random(1,4) | 4 | This works as it should |
| 1000 | 1000 | 1000 | This works as it should |

## NPC collision

Verify whether the list of highscores currently in the highscores.txt are shown as text above the NPC when the user collides with him.

Test 1:

**

The high scores are shown, but they are not very clear to the user. I also want them to output vertically like in the highscores.txt file, so although this test is successful it has brought to my attention this problem and this is what I will try to fix and check with the next test.

Test 2:

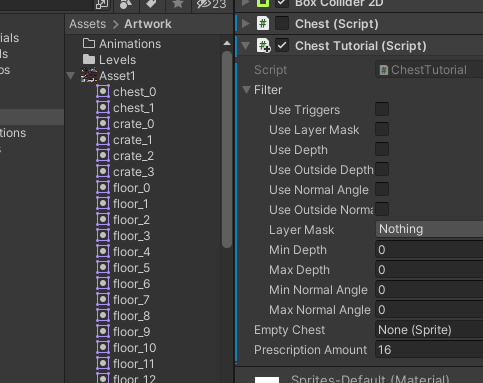


Works perfectly. The high scores are shown exactly as they are in the text file and are very clear to the user because they are vertically ordered instead of horizontally like before.

## Sprite animation - chest

Check that the sprite animation for the chest works properly and at due time - when the player ‘collides’ with it.

Test 1: works for the chest tutorial sprite, but not in the right way. The sprite completely disappears, which is not what I wanted. It turns out this issue was caused by me forgetting to add the sprite to the box:



*Look at the bottom right corner that reads Empty Chest.*

Instead of None (Sprite) it should read chest\_1 instead which is seen on the left, all that was needed was for me to drag and drop the sprite into the Empty Chest box.

The test has however made me recognise again that the player collision is not working whatsoever, in fact I’m not sure why the player doesn't just phase through the walls as well. This may be because the walls are a different z level compared to the character, but as I have used rcHit2d I am not sure if this is certainly the case. I have done tests on this, many more than the others combined, but have not recorded any. I have come to the conclusion that it is best to switch to BoxCast for collision detection.

## Objective-specific

1st Objective

The map and character design are satisfactory and up to standard as to what my peers/end users wanted. I consulted with them along the way to make sure that they would enjoy the setting, but also added bits and pieces that I thought would go well since this is my project after all. The graphics of the game can not be tested, but the teleporter and the tree can be:

(FV 0:25)Teleporter - this can be seen working as part of the final testing video that shows all the elements of the game working together.

Tree - the tree structure was used to represent the map. When the enemies in an area are defeated, the crate should effectively explode (it is hidden and the collider is disabled). I will test 3 different areas to make sure that this works properly.

| **Crate/area** | **Expected** | **Actual** | **Meaning** | **Comments** |
| --- | --- | --- | --- | --- |
| 1 | Crate to disappear from game | Crate disappears | Works | None |
| 3 | Crate to disappear from game | Crate disappears | Works | None |
| 6 | Crate to disappear from game | Crate did not disappear | Works | Under this table |

I did not expect crate 6 to not get destroyed and therefore it was quite a surprise to me. I only had one enemy connected to this crate as part of the list of enemies that need to be destroyed for this crate to disappear, but the enemy was there. I also checked the Enemy GameObject itself and then found that I forgot to add the crate that represents that area to the enemy’s health script. I did this and tested again, this time the crate did disappear.

2nd Objective

This objective has been successfully met. The character (as seen in the Sound and player movement - Test 3 video) moves properly, creates sound when he is moving, and the sound attracts enemies towards him. The amount of sound he makes is dependent on the distance covered in a time interval (that is dictated by me). I have set the value of the time interval through repetitive testing to 0.5 as I found this value to be the best. The testing for this is represented under.

| **Value of time interval** | **Expected** | **Actual** | **Meaning** | **Comments** |
| --- | --- | --- | --- | --- |
| 1 | Sound to play when character moves | Sound is playing much later than when character moves | Time interval is too long | Time interval should be reduced |
| 0.2 | Sound to play when character moves | Volume of sound is changing too often | Time interval is too short | Time interval should be increased |
| 0.5 | Sound to play when character moves | Volume is changing nicely alongside movement | Time interval value is working well with the game | None |

3rd Objective

(V&FV 1:12)The system works hand in hand with the player movement and sound system just as it should. Enemies are indeed attracted to the area that the player has made the sound in, and deal damage upon contact with the player. There exists a simple BoxCast collision detection algorithm that I created for the enemy and that ensures that the enemies do not pass through the character(walls) but collide with him (stop before passing through the walls). The tests for the enemy attraction to the area the sound has been made in can be seen within the Sound and player movement system - Test 3 video. The test for checking whether the enemy deals damage to the player can be seen in the final comprehensive testing video.

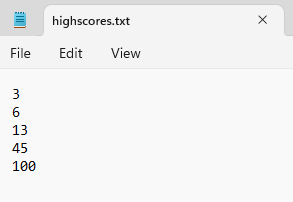
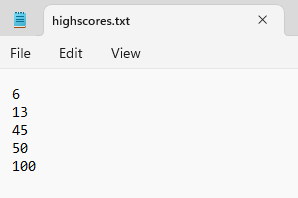
4th Objective

(V&FV 1:12)The only part of this objective that is independent of the rest of the objectives and their corresponding test in some way is increased damage to an enemy if the player backstabs them, which can be seen separate to the final video. The backstabMultiplier variable should also increase by 1 after a backstab and this is tested alongside it within the video. I will show that the enemy and player do indeed die if they lose all their health in the final testing video.

5th Objective

The tests for this objective are the following:

* (FV 0:33)Does pressing the escape key pause the game and all movement?
* (FV 1:37)Are the high scores updated and saved in highscores.txt when the player dies?
* What happens if there is no highscores.txt file already on the device?
* Are the high scores sorted correctly?

**

The image on the left is before the new high score of 50, while the image on the right is after the new high score. As seen in the images, the integers in the text file are sorted correctly.

## Final Test

*A mistake was left in the final testing video, the timeInterval was set to 0.4 instead of 0.5 and this caused the game to play the sound too often throughout the video.*

The final test video showed the components of the game working together, the enemies dying, the high scores being updated, the player dying, and the teleporter. The test for what happens if there is no highscores.txt file already present on the device was skipped over as I believe this part can be better seen as part of the technical solution.

(FV)This test was a success in most parts (other than the sound playing too often). Everything worked in conjunction with each other to produce the game, though I must emphasise that the game is much harder than it seems in the video and that it was only because I had around 1 million health for testing purposes that I was able to survive that long. My hope for the game now is that my end user(s) is satisfied with the final product and enjoys the stealth/tactical play style of the game.

# Evaluation

## Objectives

*Each number denotes an objective, and the letters depict the person giving the feedback. The letter a is for me, b is for my end user, c is for my evaluation, and d is how it could be improved.*

1. Map and character design:
   1. The design I implemented at first for this project stuck through till the end, including the player sprite and map design. This shouldn’t have happened and it instead should have changed and been improved from its original form. A GUI should have also been added to the game in my opinion as this is a small but valuable element to games in general. The positives are that the portal does work as it is supposed to, and gives me the freedom to use it further with more scenes easily. The implementation of it specifically as well makes it easy to randomly teleport the player by making the sceneName a list of strings instead of a string. The tree structure is completely integrated with the game and enemies and the recursive function to check for whether an area is cleared works correctly.
   2. My end users complained that the theme they wanted running throughout the game (dark mythical land) didn’t come out throughout the game enough and in the way they wanted. They said they wanted ‘a more complicated and better looking game’. Some of my users specified that they wanted more animations in the game, especially when backstabbing and attacking the enemies as they want to have a visual response to them pressing attack.
   3. I understand what my end user is saying completely. I did expect that some of my end users would not be satisfied with the overall design of the game from a graphics perspective but I didn’t expect it to bring this great of an effect upon the experience of the game. I should have spent more time developing the map and components within it, especially since these are small things that don’t require much time when compared to the coding of the game and creation of classes, etc.
   4. I could change the character sprite to one with relevant animations and include the running of these animations within my code or through Unity. I could also find and use a wider variety of assets to make the map more unique and have multiple of the elements my end users wanted - one specified a waterfall for example. To make use of the portal even further I could also change sceneName as I said before in part a to a list of strings and use a random generator to teleport me to a random scene, though this would require more scenes to be added to my game.
2. Character movement/sound system:
   1. This objective has been fully met out - the character can move using WASD or arrows and creates sound when they move which attracts enemies to the area they were in. I don’t think there is anything specific with this objective that I missed out on, nor do I think my end users will be disappointed with the outcome of this part of the game. I do however want to change the sound that is played as I do not think it is a good sound for the player noise creation.
   2. My end users enjoyed this part of the project the most, many of the comments I received were praise for the innovative idea that I implemented here. Some did complain that the noise got too loud at times and wasn’t being calculated properly.
   3. My end users seem to have misunderstood part of the player movement/sound system was for sound to be created passively no matter what so that the game is not too easy for the player, and this is why they felt that they were being heard unnecessarily throughout the game and they thought the noise wasn’t being calculated correctly.
   4. I could have added a noise for the footsteps the player takes, a more applicable sound to my player noise creation, and could have added special manoeuvres to the character, such as a crouch function that lets the player move but with less speed and less noise created.
3. Enemy AI mechanism:
   1. This system does work with the sound movement system properly, but just isn’t how I envisioned it. I think I should have used a more advanced pathfinding algorithm then the custom once I made such as A\*. The solution however does adhere to the requirements I set out within this objective so it is therefore a success.
   2. My end users didn’t have any complaints about this part of the game whatsoever. All of my end users said that the idea where the enemy is attracted only to the place where the player made sound made them feel like the enemies did not have eyes, which gave a creepy vibe to the game. They said that it would have been smart to implement my enemy sprite as such to reflect that, like making them bats.
   3. I am interested in the idea my end users have thought of, implementing my enemy sprites to those without eyes. I think this is a great idea and works perfectly with the player sound system I implemented story-wise. I expected my end users to have problems with the pathfinding method I used while playing the game but I did not receive any complaints from them.
   4. Since my end users are happy with the pathfinding of the enemies, I do not think this should be changed. I do however think that with the creation of my enemies such that they are visually impaired by representing them as bats or as monsters with no eyes is a great idea and that it would polish the game a little further.
4. Damage and health:
   1. This objective is also fully met through this game and was probably the one I am most proud of. I managed to use the dot product as I had set out to do initially and have a Weapon script that calculates whether the player is backstabbing the enemy or not. On the other hand, I don’t think the fact that there is only one weapon is a good choice. I can now see that I should have added multiple weapons to the character to bring variety and excitement to unlocking new areas, as opposed to only an increase in difficulty.
   2. Most (if not all) of my end users wanted more weapons in the game. The same proportion of end users however were pleased with the backstab function in the game. After consulting with them specifically about the completion of this objective, they have arrived at a similar consensus to me that there should be more weapons when an area unlocks so that they are incentivised to open up the next area and there is a fun progression of items throughout the game. My end users did also mention that they wanted a larger variety of enemies with different attacks, such as those who shoot at the position where noise is being made instead of only going to that position.
   3. I don’t think I can agree with the enthusiasm with which my users wanted more weapons and the number they wanted, but I agree with their point. I find it especially interesting that they want an enemy that shoots at the area where sound was made, and think that this is a great idea to look into.
   4. From my end user and my perspective, I believe that I should add more weapons and create different enemies, as this will bring improvement to their total satisfaction of the game and their interest in games which have similar genres to this/playstyles to this.
5. Pause and save function for high scores:
   1. This works as I imagined it would, and when colliding with the NPC the list is indeed shown. The pause option also works, but I know that I should have added a little PAUSED text to the screen when the user presses pause and am not very happy with the outcome in total of this specific part.
   2. My end users complained about not knowing when they are paused or not due to the fact the game has no text or icon or menu representing that the game has been paused. They enjoyed the idea of the high scores being shown upon collision with the NPC and said “I’ve never seen something like this before”.
   3. I think my end users are right again in this case, as it just isn’t correct for there to be a pause option without a pause screen or some icon being shown. On the other hand, I am pleased that they are happy with the high scores being shown the way they are.
   4. Simply put when pause is pressed (the escape key) a window of sorts should pop up. I can do this by creating a gameobject called Pause Menu and setting a child of it that is a Text component and have the gameobject set to true when the isPaused attribute of my Pause class is true.

## Solution as a whole

The solution only partially solved the problem my end users were having. Just about more than half of my end users I consulted with were intrigued by the ideas and elements that were part of this game, and were interested to see if there were any games that had the same ideas coherent throughout them. However, some of my end users didn’t feel this way at all and they each reported that the game was too aesthetically displeasing/boring for them to play it. Seeing as multiple of my end users reported the same fact to me, I believe that this is a core component of the game that I overlooked. This component of the game was one I was enthusiastic about during the beginning of development of the project/game, but one that became too boring as time went on. I think that I didn’t take into account the importance of this element in my game (and games in general), and took somewhat for granted what other games have compared to mine.

Nonetheless, the majority of my end users reported that they enjoyed the game and felt “the game is pretty basic actually, but it's nice how you have to be really careful around enemies and take your time planning your route and stuff…”.

I did also have end users wondering whether there were any games like this one but that were more polished/refined, and I was happy to recommend some. Overall, I believe that this project was mostly a success, but also that if more time was spent developing the graphics and animations throughout the game my end users could have maybe enjoyed the game more.

# References

<https://www.youtube.com/watch?v=b8YUfee_pzc>

*This video is the one which any of the classes or script refers to like Collecting - R.*

3/7/2022

<https://www.lucidchart.com/pages/flowchart-symbols-meaning-explained>

5/3/2023

<https://www.conceptdraw.com/How-To-Guide/flow-chart-symbols>

5/3/2023

<https://www.mathsisfun.com/algebra/vectors-dot-product.html>

5/3/2023

<https://stackoverflow.com/questions/463155/how-does-the-ternary-operator-work>

1/3/2023

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

1/3/2023

<https://0x72.itch.io/16x16-dungeon-tileset>

*This is where all the assets I used in my game originate from.*

5/7/2022