Proposed Level of Achievement

Project Apollo 11

Team Information

1. Team Name:
   Rogue

2. Team ID:
   6282

3. Team Members:
   Sun Siliang
   Chen Siyu

4. Team Background:
   Our team has a good coding base for the mainstream coding language including python, java, and javascript as we have gone extra miles to enrol in online courses for Python and Java prior to university admission. As for academic performance, our team has an average CAP of 4.90. with A+ in CS1010 and A in other maths and CS modules. We believe that our team can successfully end up with a high-quality application.

Project Scope

Our project aims to create a rogue-lite video game for gamers, using randomly generated contents to provide infinite possibilities to explore.
Ordinary games usually have a fixed set of maps and levels. Players find the games boring after exploring every map/level. Our game, "Neverending", is a 2D rogue-lite, single-player, offline game which offers players an immersive experience in exploring a world of ever-changing terrains and various kinds of enemies. It adopts a procedural generation algorithm so that every game, every level will be different and new, so players can play our game again and again to explore infinite possibilities, and there is no single formula to win the game.

Motivation

In recent years, a game called Dead Cells went viral with the rise of PC end gaming platform Steam. After having immersed ourselves in the game for tens of hours, we realised how the use of procedural generation plays a significant role in creating a dynamic and re-playable gaming experience for roguelike games. As gamers ourselves, we are passionate about exploring the dynamic interplay between procedural generation and handcrafted design, creating a game that feels fresh yet challenging with every playthrough. Moreover, we have also noticed that highly challenging gameplay, which has almost become an iconic feature of roguelike games, can be a barrier that prevents casual gamers from enjoying the game. Therefore, while drawing inspirations from the successes and innovations of "Dead Cells", such as its hybrid level design approach, we also strive to push the boundaries of procedural content generation further, exploring new ways to blend algorithmic complexity with our own creative design to offer a rich, unpredictable, and rewarding player experience.

Credits

Special thanks to all the people who have provided assistance in one way or another. Their guidance and code examples were crucial for our learning of Unity and C#, as well as the development of the game. Here is the list of resources that we have used or learnt from:

1. Multi-Weapon system by Bardent:
   https://github.com/Bardent/Weapon-System-Tutorial-Series-Unity
   www.youtube.com/@Bardent
2. 2D top down game tutorial by AbdelmoumeneUnity:
   http://www.youtube.com/@AbdelmoumeneUnity
3. Learn how to make 2D platformer by Chris's tutorial:
   http://www.youtube.com/@ChrisTutorialsYT
4. Procedurally generated maps tutorial by Sunny Valley Studio:
   https://www.youtube.com/watch?v=-QOCX6SVFsk&list=PLcRSafycjWFenI87z7uZHFv6cUG2Tzu9v
5. Arts by
   - ➔ Sven Thole:
     sventhole.artstation.com
   - ➔ Clembod:
     https://www.artstation.com/clembod
   - ➔ Luiz Melo:
     https://luizmelo.artstation.com/
   - ➔ Karsiori:
     https://assetstore.unity.com/packages/2d/pixel-art-key-pack-animated-271986
   - ➔ Mushra Games:
     https://assetstore.unity.com/packages/2d/environments/brown-rock-tileset-top-down-32x32-pixel-art-281170
   - ➔ Krishna Palacio:
     https://assetstore.unity.com/packages/2d/environments/minifantasy-dungeon-206693
   - ➔ Szadi Art.:
     https://assetstore.unity.com/packages/2d/environments/rogue-fantasy-castle-164725
   - ➔ Cainos:
     https://assetstore.unity.com/packages/2d/environments/pixel-art-top-down-basic-187605
   - ➔ Mk.matheusklein:
     https://assetstore.unity.com/packages/audio/music/free-casual-music-pack-242591
   - ➔ War:
     https://assetstore.unity.com/packages/2d/characters/slime-enemy-pixel-art-228568

Design details

- Background story:

Our protagonist is a prisoner who comes from the bottom of the society. He was accidentally given the ability of resurrection, which allows him to resurrect in the dungeon everytime he dies. Having suffered so much as a prisoner, he decided to overthrow the dictatorship that is ruling over the country with his ability. After the player has killed the tyranny countless attempts, he became the king and the owner of the castle. However, not long after, he himself turned into a tyranny just like the previous king he killed, and has gradually lost his ability. Without his knowing, there is another soul in the dungeon who has also been given the same ability of resurrection and decided to kill him to claim the throne.

- Rationale:

As we are somehow bound by the mechanism of Roguelike games, we created the above game settings to fit the gameflow. Our game comes in cycles and there is no one ending point, which is the reason why we

named it Neverending. After the final boss fight, the player will be sent right back to the starting point and he can start a new game. This feature is also depicted in the background story where there is a constant cycle of ruling and rebellion, much like the case of monarchy in history. Therefore, as a countermeasure to the unavoidable repetitiveness in the gameflow, and the limited number of levels we can create in one single game, we relied on procedural generation to randomly generate maps, enemies and rewards, such that every experience is unique for our players. There is also the characteristic permadeath setting in our game, which is also furnished into the ability of resurrection, but limited to the dungeon.

User Stories

1. As roguelike game lovers, playing on the same map repeatedly can be quite boring. We want to have different maps and enemies so that we get fresh experiences every time we play.
2. As roguelike game lovers, we want to have permadeath mechanisms so that the gameplay becomes more challenging and interesting with irreversible penalties.
3. As gamers, we want to have random rewards so that we get a sense of surprise and satisfaction when we have no idea what we will get after a tough fight.
4. As gamers, we want to see nice aesthetic designs of characters, items and scenes so that the gameplay will be more appealing to us.
5. As casual gamers, we want to have a simple and easy-to-learn system to control and navigate in the game so that we can get proficient without spending too much time practising.
6. As gamers, we want the game to have an increasing difficulty and to have no end so that we can keep challenging ourselves to get better skills and score better records.

Features

GamePlay Features

1. Introduction to the game
   Upon starting a new game, the player will enter a level. To pass a level, the player needs to collect all keys (the total number of which is dependent on the size of the map) and arrive at the Portal. Each of the keys is held by a random, anonymous enemy, and the player has to defeat it to get the key. In addition, our game uses the permadeath mechanism, i.e. he/she has only one life. Death of the player marks the end of the level and the game. Upon successfully completing a level, a new and more challenging level will be generated. This process will loop until the player dies or quits.

The Key

Enemy Drops a Key after Defeated

All Keys Collected to Enter the Next Level (Highlighted)

2. Player movement

The layout of our game is top-down. Under this layout, the player only has movements in the up, left, down and right directions, which are controlled using 'W', 'A', 'S', and 'D' keys respectively, at a uniform speed. Player movements are animated. When movement keys are pressed, the Player sprite will appear running in that direction.

Player idles

Player runs

3. Combat

Player attacks with the primary weapon by pressing the mouse left button, and attacks with the secondary weapon by pressing the mouse right button. The attack speed is limited by attack animations and attack interval. If attacks are spammed, excessive attacks will be invalid and not able to deal any damage. The direction that the player is attacking is dependent on where the cursor is. For ranged attack, the angle at which the projectile is shooting out is set within 45 degrees above and under the horizontal line to be more realistic.

Player uses a melee attack

Player uses a ranged attack

4. Weapons

   In our game, the player can switch weapons easily. by pressing “Q”, the player can switch primary weapon and by pressing “E” player can switch secondary weapon. We have designed various types of weapons so that our players have more choices in battles.

   a. Excalibur, a legendary sword, is slow and heavy but delivers immense damage. Perfect for those who prioritise power over speed, it symbolises ultimate battlefield strength.

Statistics:

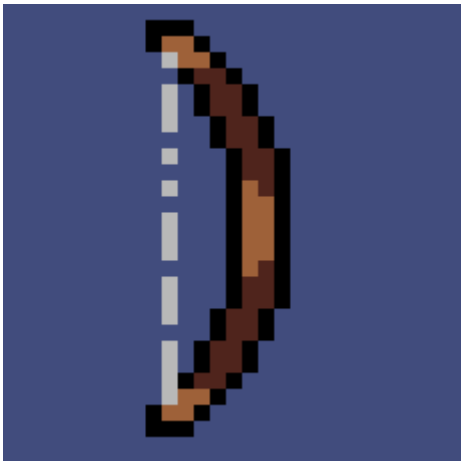
- ➔ First Attack
  Damage: 10 HitBox 1.4 * 1
- ➔ Second Attack
  Damage: 10 HitBox 1.4 * 1
- ➔ Third Attack:
  Damage: 15 HitBox 1.2 * 2

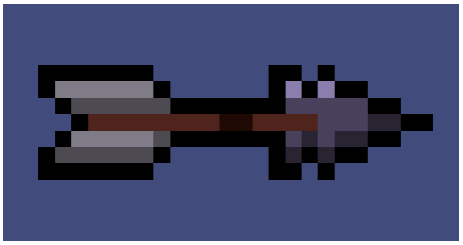
b. The Scarlet Blade is a swift, precise weapon known for its crimson arcs. Lightweight and agile, it delivers rapid, powerful strikes, making it perfect for warriors who value speed and finesse in combat.

Statistics:

- ➔ First Attack
  Damage: 10 HitBox 1.4 * 1
- ➔ Second Attack
  Damage: 10 HitBox 1.4 * 1
- ➔ Third Attack:
  Damage: 15 HitBox 1.2 * 2

c. The Ironwood Bow, a powerful, durable weapon known for its strength and precision. Ideal for archers who value reliability, it delivers powerful, accurate shots, symbolising unwavering strength in combat. The specially crafted Black Iron Arrow can easily penetrate the hard skin of a monster, but also heavy armour, making it the best company for the Ironwood Bow.

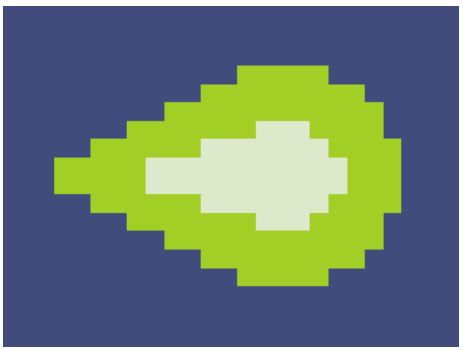




Statistics:

- ➔ Attack Damage: 10
- ➔ Attack Range: 15

d. The Necromancer's Tome, filled with dark and forbidden knowledge, empowers its wielder to command the undead, to overwhelm and terrify its enemies by summoning the forces of the wraith. The Necron Missile that the Necromancer's Tome is casting can burn through any surface with the fire of wraith and cause heavy damage to whoever it lands on.



Statistics:

- ➔ Attack Damage: 10
- ➔ Attack Range: 15

5. Enemies

For the enemy's battle system, we have designed it in such a way that the attack action hit box is only activated when the strike is about to land. So the player will not be taking damage even if the enemy swings the sword, which allows More room for manoeuvre. We have also designed unique combat systems for different bosses, so as to make the game more diverse and interesting for the player.

a. Light Bandit

Light Bandits patrol around their locations. If the player goes near them, they will be alerted and start chasing and attacking the player using their swords, until the player flees or perishes.

Statistics:

- ➔ Melee Attack: 15
- ➔ Melee Attack Range: 1
- ➔ Melee Attack Rate: 1
- ➔ Defense: 5
- ➔ Health: 30
- ➔ Speed: 0.8

A Light Bandit

b. Heavy Bandit

They are similar to Light Bandits, but stronger and better equipped.

Statistics:

- ➔ Melee Attack: 18
- ➔ Melee Attack Range: 1
- ➔ Melee Attack Rate: 1
- ➔ Defense: 20
- ➔ Health: 50
- ➔ Speed: 0.7

A Heavy Bandit

c. Blue Slime

Blue Slimes are timid monsters. They do not attack anyone actively. They do not even move. However, if the player gets too close, they will deal damage to him/her to defend itself.

Statistics:

- ➔ Melee Attack: 20
- ➔ Melee Attack Range: 0 (upon contact)
- ➔ Melee Attack Interval: 2
- ➔ Defense: 0
- ➔ Health: 40
- ➔ Speed: 0 (stationary)

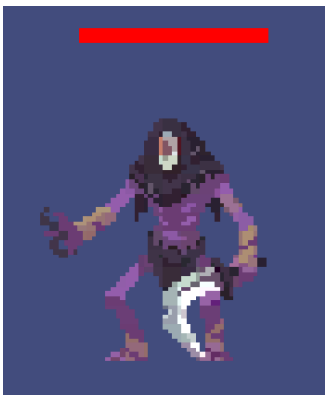
A Blue Slime

d. DeathBringer

As its name suggests, the DeathBringer brings death. It is a huge and intimidating creature wielding a terrifying sickle that reaps lives. Once the player enters its territory, it will hunt him/her down. The DeathBringer is capable of summoning Dark energy to strike its enemies from distance. The ranged attack is swift and deals immense damage, but it has a very long attack cooldown. When its ranged attack is on cooldown, it will move towards the player and make powerful attacks with its giant Scimitar. Players need to watch out for this dangerous boss because a heavy price will be paid if they make a mistake!

Statistics:

- ➔ Melee Attack: 30
- ➔ Melee Attack Range: 1.5
- ➔ Melee Attack Rate: 1.5
- ➔ Ranged Attack: 50
- ➔ Ranged Attack Range: 5
- ➔ Ranged Attack Rate: 12
- ➔ Defense: 30
- ➔ Health: 100
- ➔ Speed: 1.5



DeathBringer

e. Flame Master

By transforming his body into a stove, The Flame Master suffers great pain in exchange for the formidable powers of the fire. If he senses the player, he will fly to him/her and unleash his fire at him/her, until his body cools down and he finally can enjoy a moment of peace, just a moment.

Statistics:

- ➔ Melee Attack:
- ➔ Melee Attack Range:
- ➔ Melee Attack Rate:
- ➔ Defense:
- ➔ Health:
- ➔ Speed:

An Evil Wizard

f. Demonic Inquisitor

Demonic Inquisitors do their jobs efficiently. They skip inquisition and jump to execution. They wield a strange weapon at their subjects, who are immediately turned into dust, leaving that weapon a horrifying mystery. He attacks by wielding its staff-like weapon and summoning a horrifying skull to attack its enemies.

Not only does he have huge attack range and high damage, Demonic Inquisitor also possesses high movement speed, making him one of the most troublesome bosses.

Statistics:

- ➔ Melee Attack:
- ➔ Melee Attack Range:
- ➔ Melee Attack Rate:
- ➔ Ranged Attack:
- ➔ Ranged Attack Range:
- ➔ Ranged Attack Rate:
- ➔ Defense:
- ➔ Health:
- ➔ Speed:

A Demonic Inquisitor

g. Guard Sorceress

These evil sorceresses surrender their minds to the demons to become their immortal guard. The Guard Sorceress cast guided missiles at any one who enters her guarding area. So if some spells come from nowhere and attack you, you will know it's the Guard Sorceress.

Statistics:

- ➔ Melee Attack:
- ➔ Melee Attack Range:
- ➔ Melee Attack Rate:
- ➔ Ranged Attack:
- ➔ Ranged Attack Range:
- ➔ Ranged Attack Rate:
- ➔ Defense:
- ➔ Health:
- ➔ Speed:

A Guard Sorceress

6. Infinite Levels

During MS3, We have decided to change our game to endless mode, as compared to a finite number of levels with different difficulty in the traditional rogue-lite games. Our main rationale is that it is more interesting for players to find out how far they can go in one attempt compared to only a limited number of levels. In addition, balancing is tricky as it requires a lot of actual testing to find the sweet spot between too easy and overly challenging, and even with the same settings, the actual difficulty is subjective to different players. As a result, we have designed the levels in a manner where enemies are spawned with higher stats as the number of levels conquered increases. Players will need to pick up permanent buffs to equip themselves so as to survive

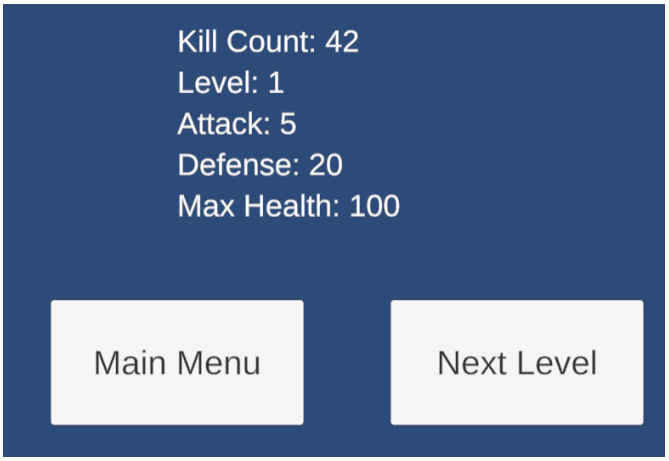
longer.

7. Record System

During the game, there is a real time recorder which records the level and kill count (the number of enemies defeated using this life). This recorder will keep recording and updating, as long as the player stays alive.

Recorder (highlighted)

After passing a level, in the Transition Scene, the player's statistics will be shown.



Statistics in the Transition Scene

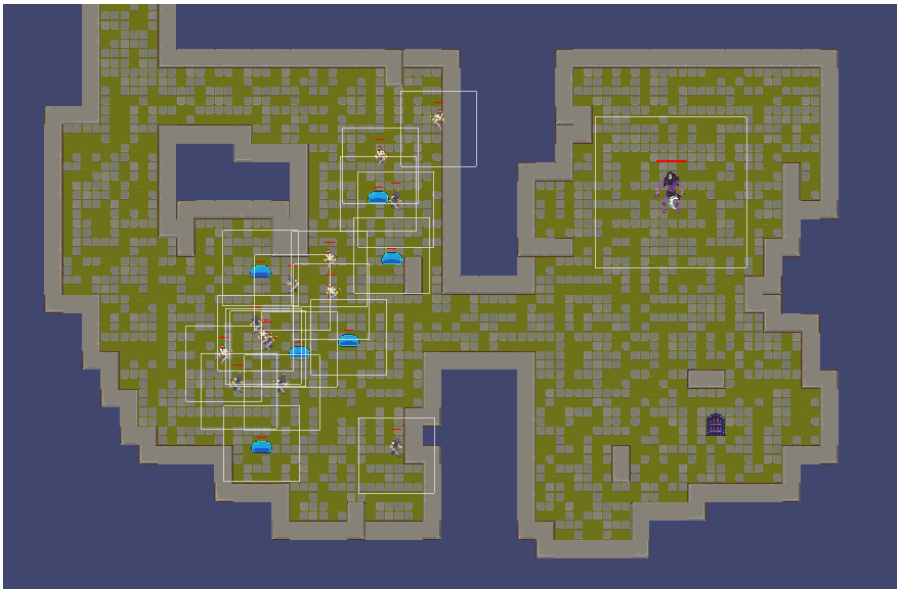
After the player is dead, in the End Scene, a summary table will be displayed, showing 3 sets of statistics, (1) statistics of the current life, (2) statistics of the highest level record, and (3) statistics of the most kill count record. If the current level or kill count breaks the record, then it will be marked red, and the current statistics will replace the record just broken.

Statistics in the End Game Scene

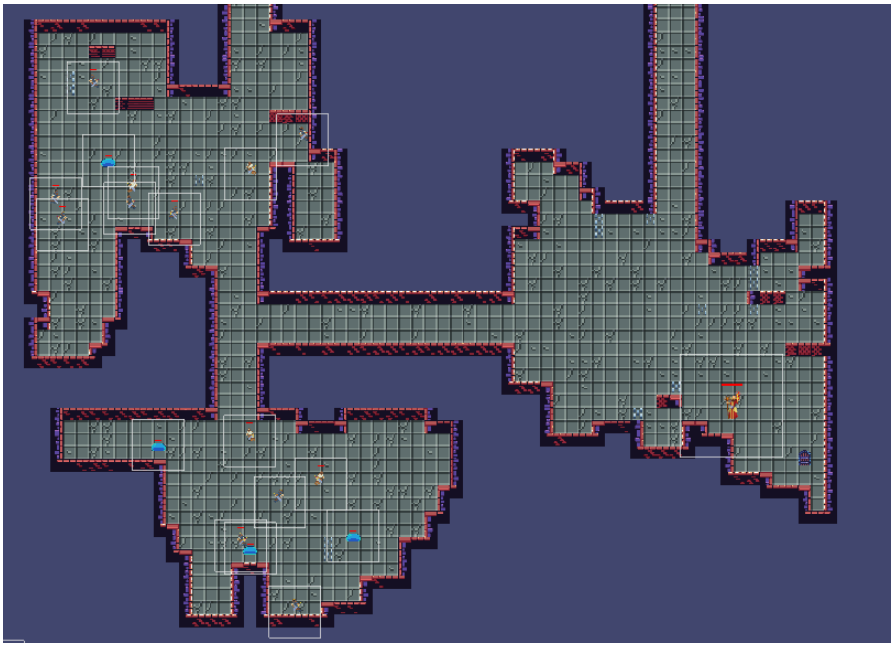
8. Random Maps and Enemies

Maps are randomly generated using a modified simple random walk algorithm. This algorithm allows us to create rooms of irregular shapes and sizes. These rooms are connected by random corridors. The corridors are widened to prevent the player and enemies from being stuck.

Besides the shapes of the maps, the art styles of the maps are also randomly assigned from a pool of art styles.



A Random Map



A Random Map

Enemies and other game objects are also randomly spawned inside the map. Most of the rooms consist of a bunch of enemies, the number of which depends on the current level (difficulty) as well as the size of the room. As the level increases, more enemies will be spawned in the rooms, including the bosses.

Enemies will drop random rewards after they are defeated, mainly buffs that increase the player's abilities. A small number of the enemies hold keys, which will drop and be collectable along with the rewards. The only way to find out whether an enemy holds a key is to defeat it and look at the loot dropped.

9. Weapon Pickup:
   We have designed a single interactable object that is able to contain weapon data we create for different weapons. When weapon pickup is within proximity of the player, interactable detector child object of the player detects the interactable object and extracts weapon data from the weapon pickup while exporting its current weapon data back to it.

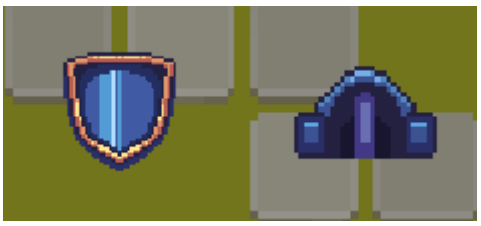
10. Buffs:
    In our game, there are a variety of collectable buffs. In general, green buffs recover health, red buffs increase attack, and blue buffs strengthen defense.
    There are two types of buffs, permanent buffs and temporary buffs. Permanent buffs increase the player's ability permanently, while the effects of temporary buffs last for a few seconds only.

Health Recovery Buffs (+10, +15)

Max Health Buff (+5)

Attack Buff (+0.5)

Attack Boost (+10 in 5 seconds)



Defense Buffs (+0.8, +0.5)

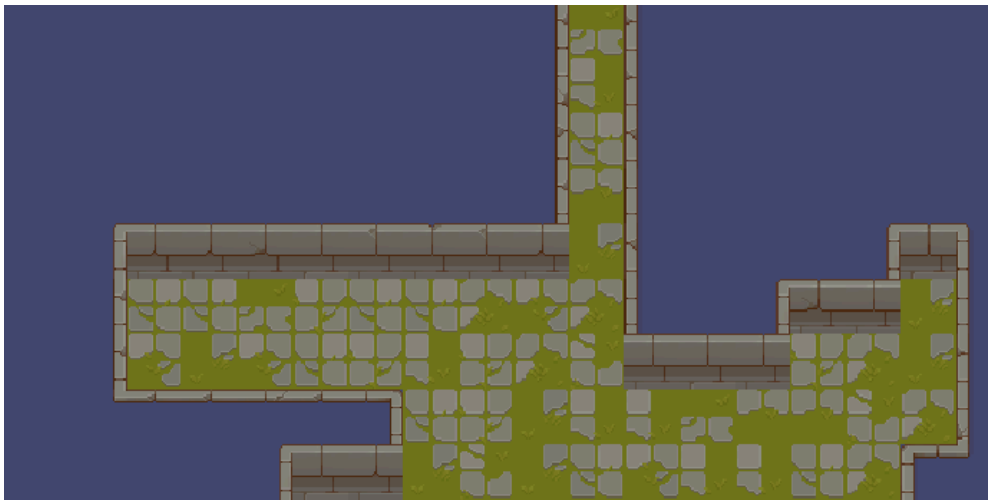
Defense Boost (+10 in 10 seconds)

Art Features

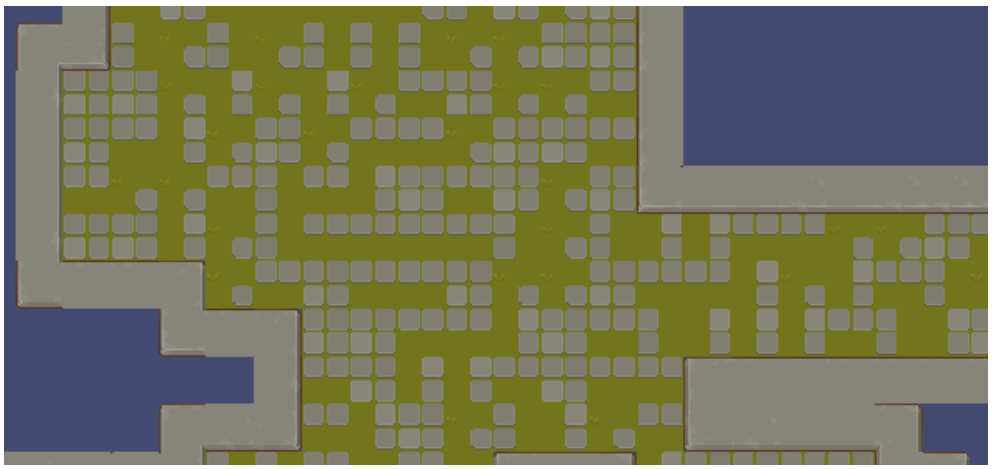
1. BGM and Music Settings
   Our game has background music (BGM). Upon opening the game, the BGM automatically starts playing. Its volume can be adjusted both at the main menu as well as in game when the game is paused. Upon quitting the game, any changes in the settings will be saved, and be loaded next time when the game is opened.

   picture

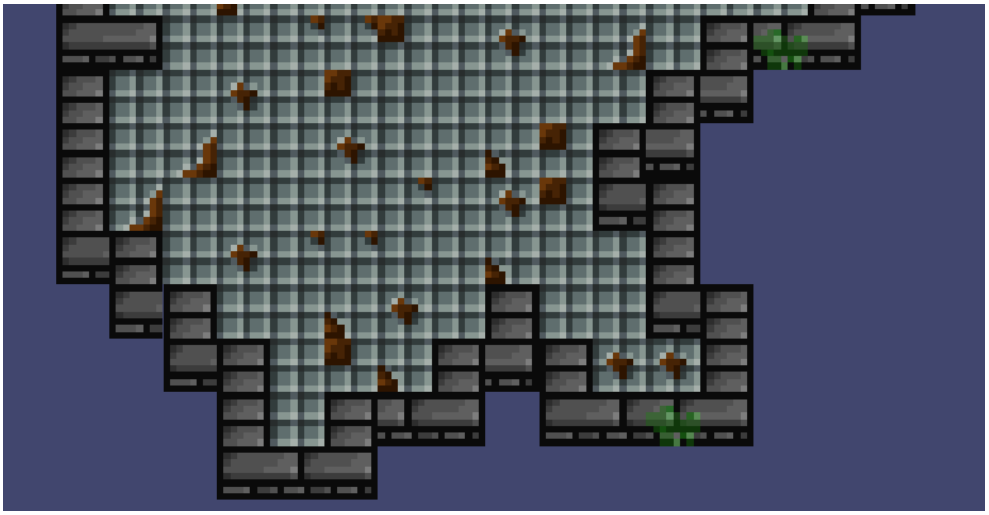
2. Map
   Our game has a total of 4 different art styles for maps: Ancient Ruin, Secret Garden, Abandoned Stronghold and Demonic Dungeon. Each game level will apply one of these styles randomly to give the player a different experience from the previous one.



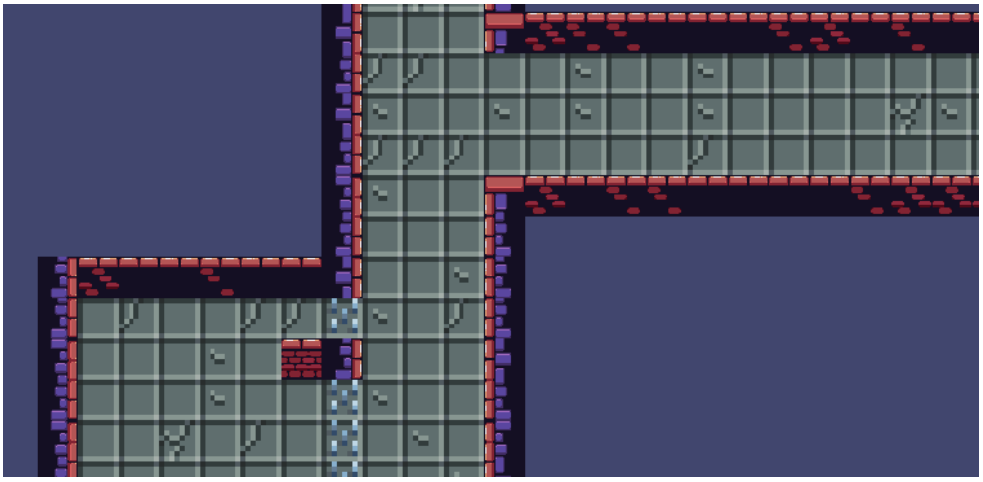
Style 1: Ancient Ruin



Style 2: Secret Garden



Style 3: Abandoned Stronghold



Style 4: Demonic Dungeon

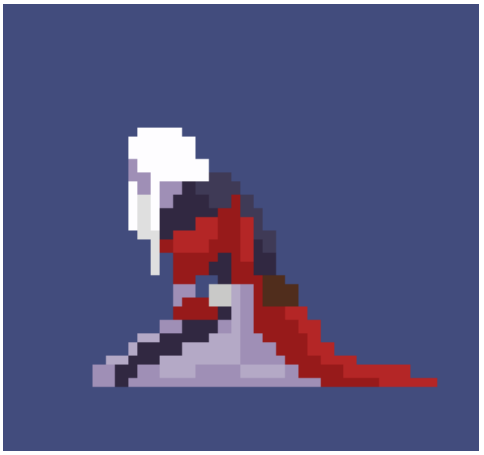
3. Animations

   Animations are widely applied in our game with details. All character actions, including but not limited to idling, running, melee attacking, ranged attacking, being hurt and being defeated, are animated. Below are some examples.

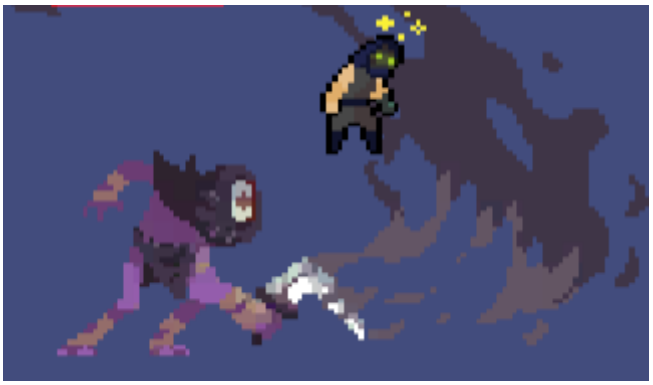
   Besides, each weapon presents a unique animation when the player uses it. This makes our

weapons distinct not only in functions, but also in visual effects.

Player is damaged by a Bandit



A Sorceress is defeated



Player is damaged by the Bringer of Death

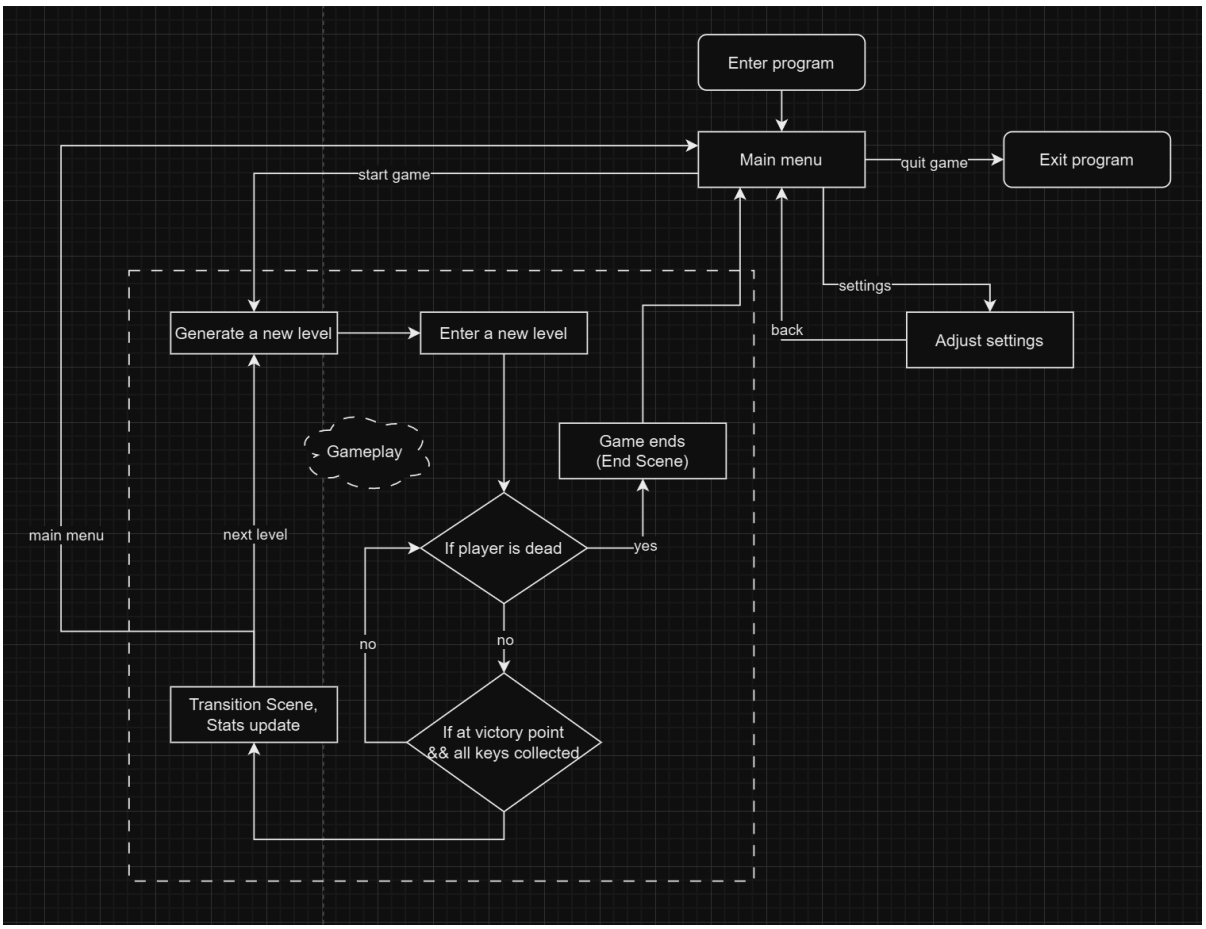
Inquisitor Going to Attack

Architecture

1. Flowcharts
   a. Overview

      The flowchart below summarises the basic gameplay. Main menu presents buttons to navigate to different functions, including “Start Game”, “Settings” and “Exit”. “Start Game” leads to a new game, where a fresh new level will be generated. “Settings” leads to a window where the player can adjust the sound-related settings. “Exit” leads to quitting the game.

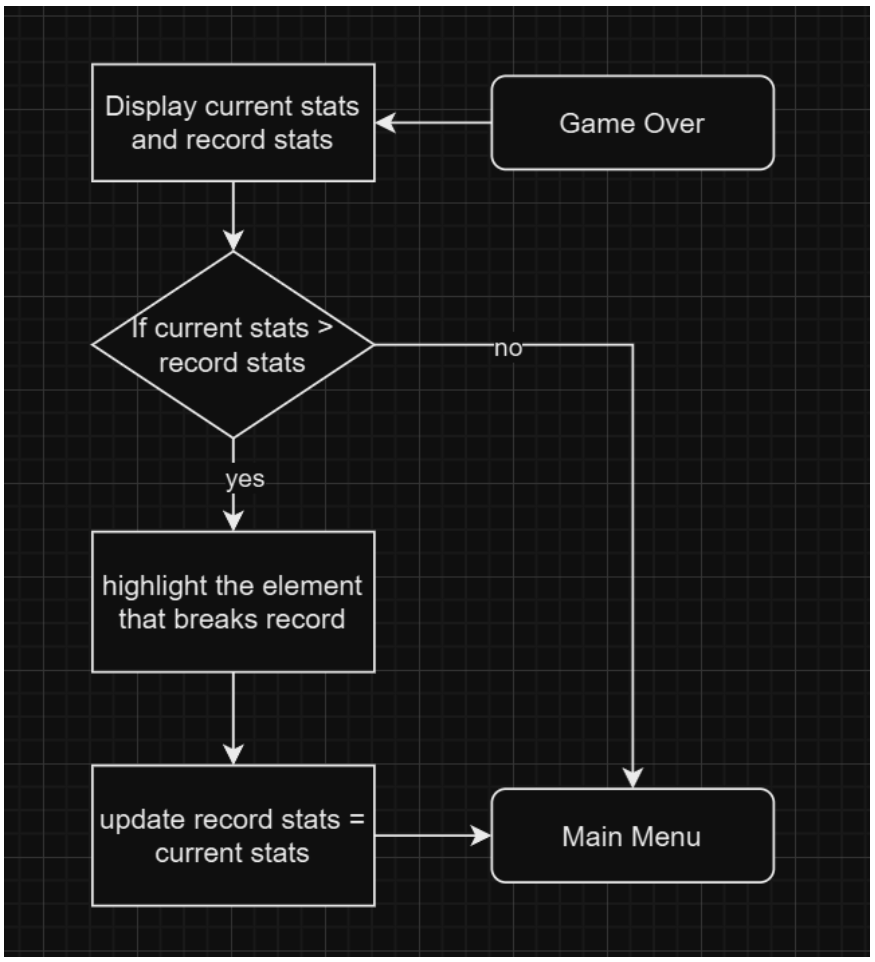
      During the gameplay, a level ends only if (1) the quit button is clicked, (2) the player dies, or (3) this level is passed. For cases (1) and (2), the player is navigated back to the Main Menu or the End Scene. For case (3), a new next level will be generated and the player will be sent there.



Overview Flowchart

b. Best Record System

The flowchart below summarises how the player's records are managed.

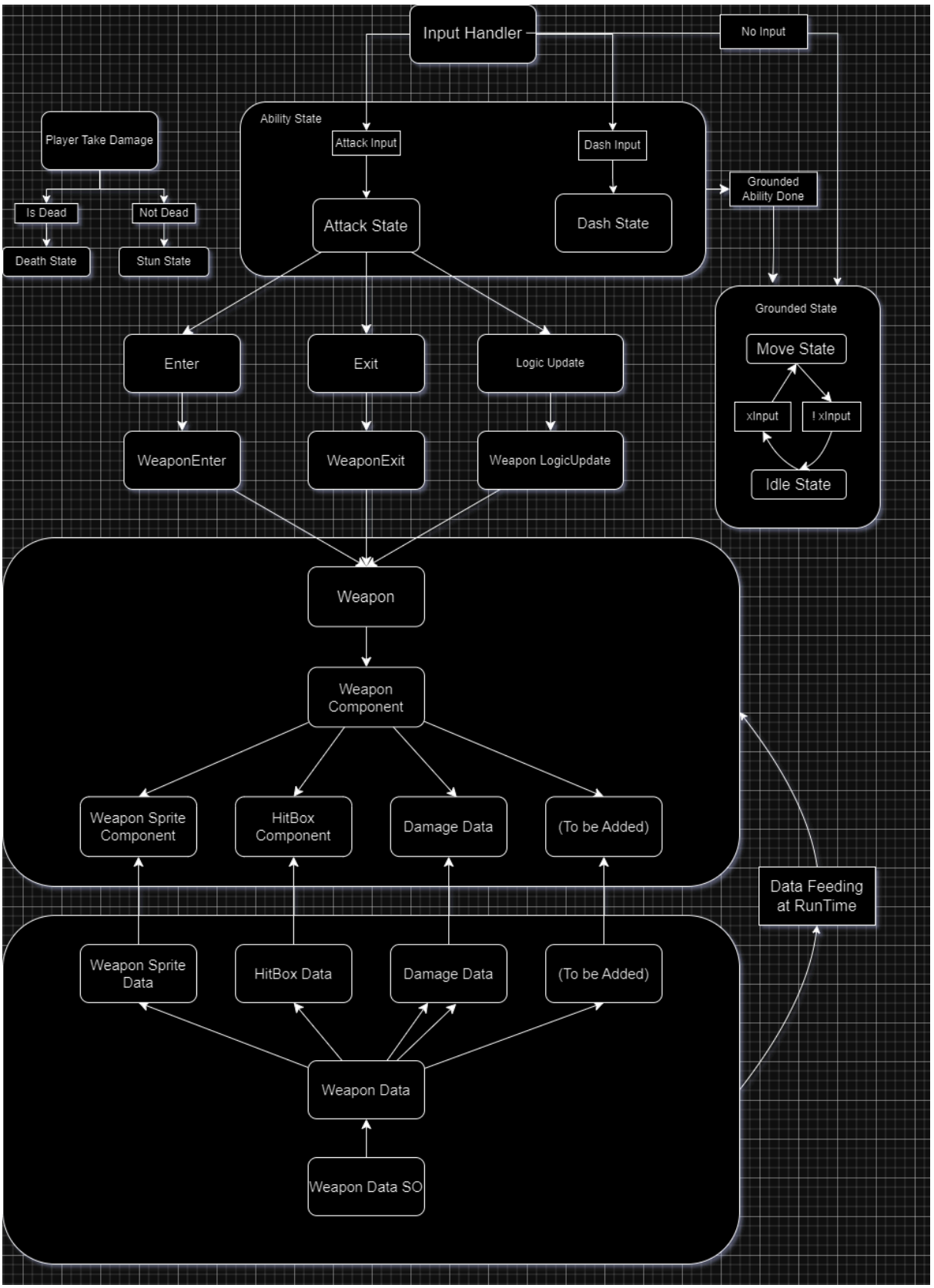


Best Record Flowchart

2. Class Diagrams
   a. Controller Player (player in short):

      For the player control section in our game, we adopted state machines to achieve smooth transitions between actions. As shown in the flowchart below, after the Unity input system receives commands from users, it will trigger functions written in the overarching states so that the player will enter the next state accordingly.

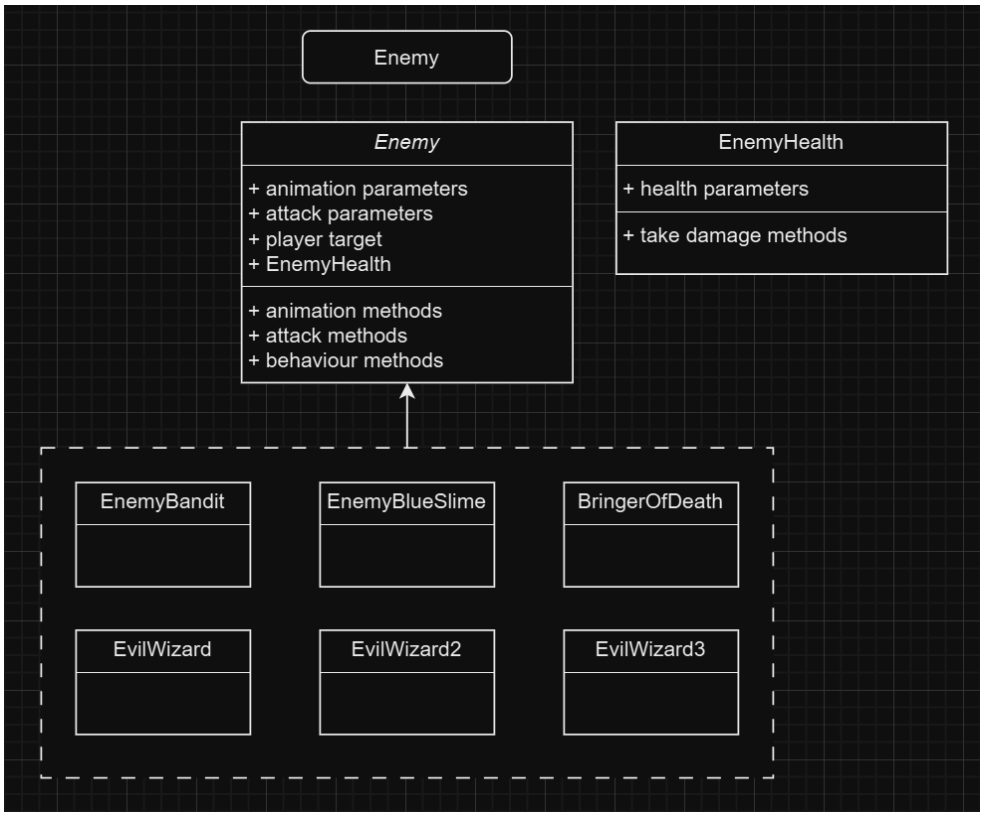
      We have also built a multi weapon system to support our flexible and diverse battle system. After the player enters the attack state after combat input like primary and secondary attack, the weapon script will be triggered. In MS2, we have built a new player which is more flexible in terms of weapons and combat. If we take a quick look at the flowchart, we can easily notice that we have separated weapons into two sections: component and data. Components include weapon sprites(animation), range (hitbox), damage etc, which are the common features we use to differentiate different weapons. There is also weapon data, which are read from scriptable objects and can be fed into respective components. Last but not least, we've also built a weapon generator, to set the weapon components based on the data input at runtime. As a result, with this system, the player can pick up or switch weapons during gameplay. In MS3, we have added two new states, stun and death. Stun states are entered when the player is taking damage, it is also accompanied by a decrease in health. a player enters death states if its health falls below 0 after taking damage, and it also signals the end of the attempt.



b. Enemy and EnemyHealth:

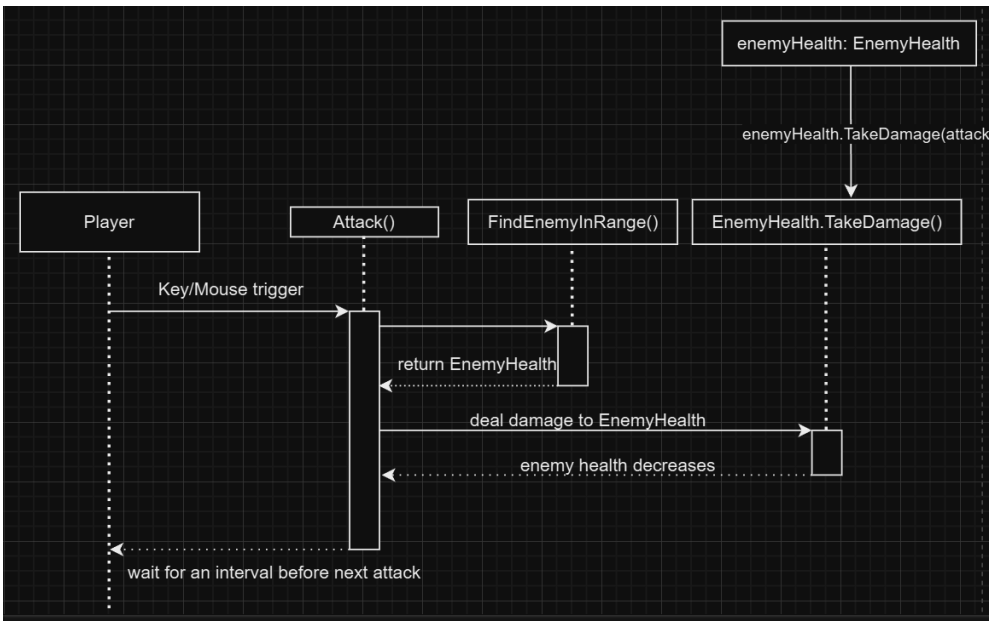
Class Enemy manages an enemy character's behaviours through parameters and methods such as float attackRange, Attack(), DropItems(), etc.. For different kinds of enemies, their special active behaviours, mainly attack related methods, are managed by the child classes of Enemy

shown in the diagram below. For example, EnmeyBandit can patrol around the map through a special method RandomPatrol(), EnemyBlueSlime deals damage upon contact (collision) through a special method OnCollisionEnter2D(), and BringerOfDeath can use ranged attack through a special method RangedAttack().
EnemyHealth mainly manages an enemy character's passive behaviours through parameters and methods such as float health, Image healthBar, TakeDamage(), etc.. All enemies use EnemyHealth. The only differences are in their health amount and defence amount.



Enemy Classes

Below shows how the Player deals damage to the Enemy. Enemy deals damage to Player in a similar way.

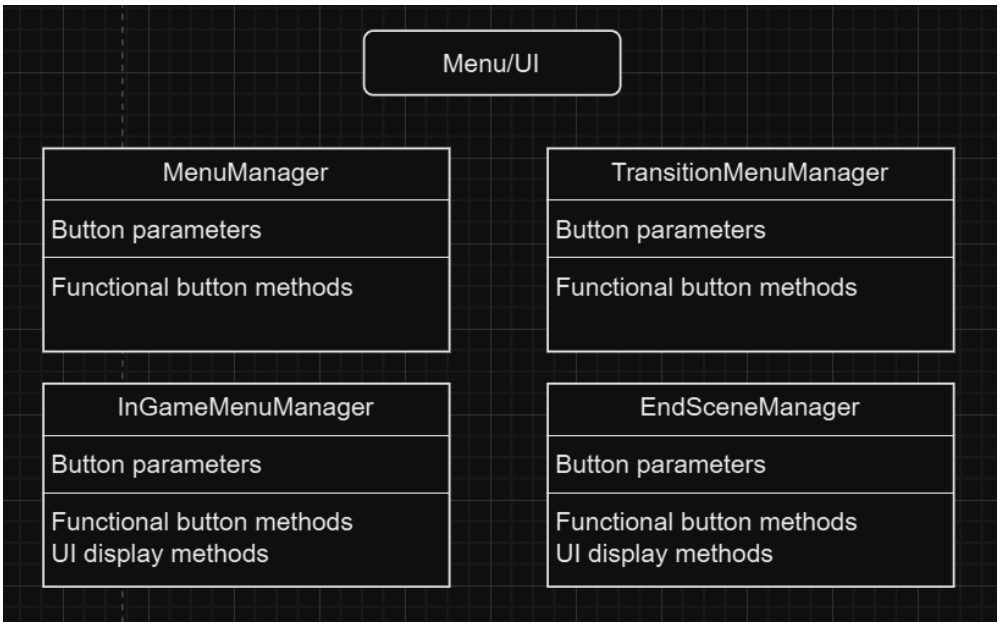


Flow of a Valid Attack by a Player on an Enemy

c. Menu/UI Managers:

Menu/UI Manager classes manage most of the UIs and displays, including buttons, sliders, symbols and signs in a scene. In each scene, a set of buttons are attached to its menu, and for each button, a function is created and attached to its trigger through Unity Inspector.

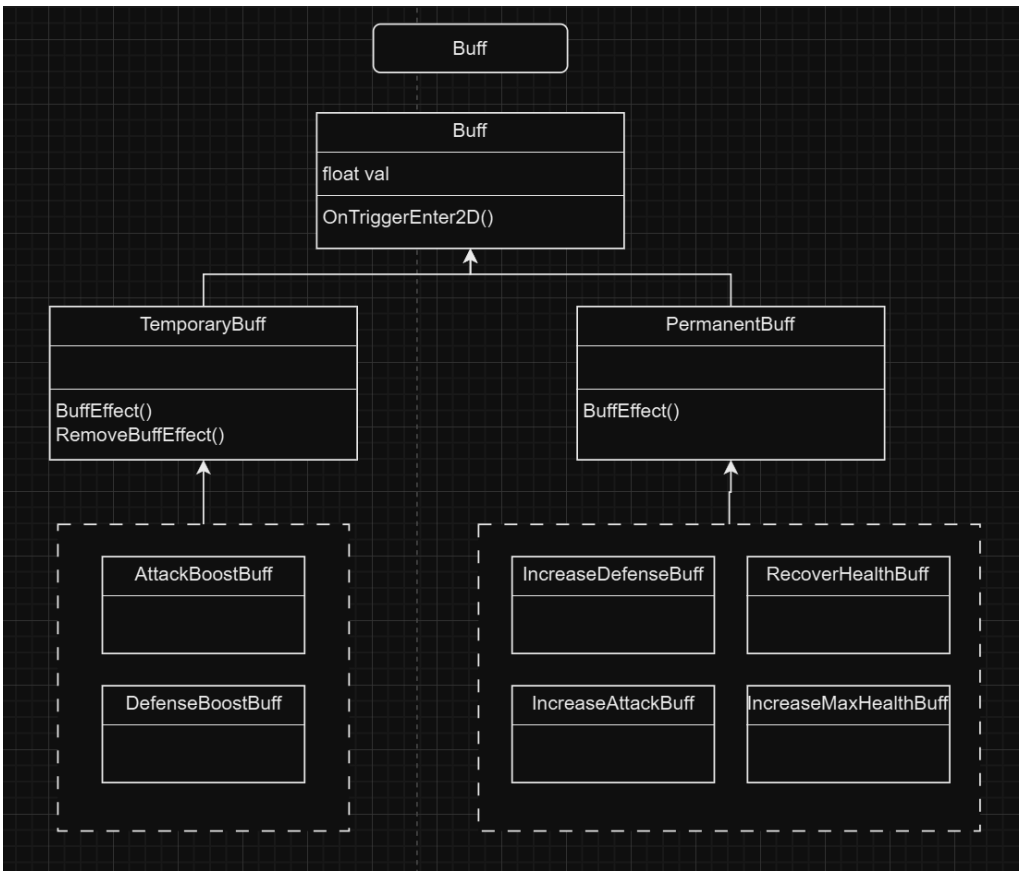
The menu classes are simple, and their purposes are very specific and dependent on their game scenes. Hence, these classes are independent to each other, without having any parent-child relation. The name “Manager” is entitled to each of them only for documentation purposes.



Menu/UI Manager Classes

d. Buffs:

There are two types of buffs, PermanentBuffs and TemporaryBuffs. The effects of temporary buffs will disappear after a few seconds. Under them, there are a number of buffs that deliver various effects.



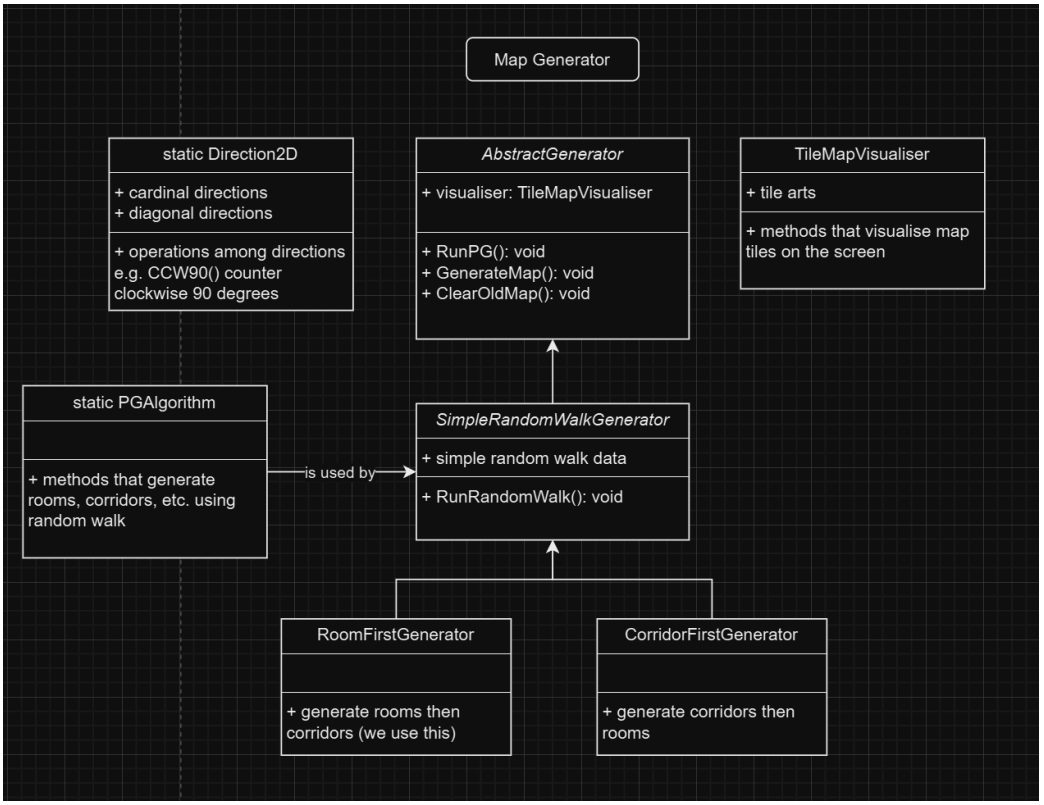
Buff Classes

e. Map Generators:

The maps are randomly generated using a random walk algorithm. As there are different procedures to generate maps, we have created AbstractGenerator as the abstract base class. The algorithm used to generate the map is the simple random walk method (walk from a starting point in random directions for many times, and the 'step prints' will be the map floor). This algorithm is stored in a public static class, PGAlgorithm.

We have mainly explored two different procedures, to generate the corridors before generating the rooms, through the class CorridorFirstGenerator, and to generate the rooms before generating the corridors, through the class RoomFirstGenerator.

We opted for the room-first way, because it will be easier for us to segregate the rooms before they are connected to each other. We need to store these rooms inside a list and sort them into the spawn room, ordinary rooms and the victory-point rooms. In comparison, the corridor-first way makes it complicated to accurately segregate the entire map into rooms without overlap.



Map Generator Classes

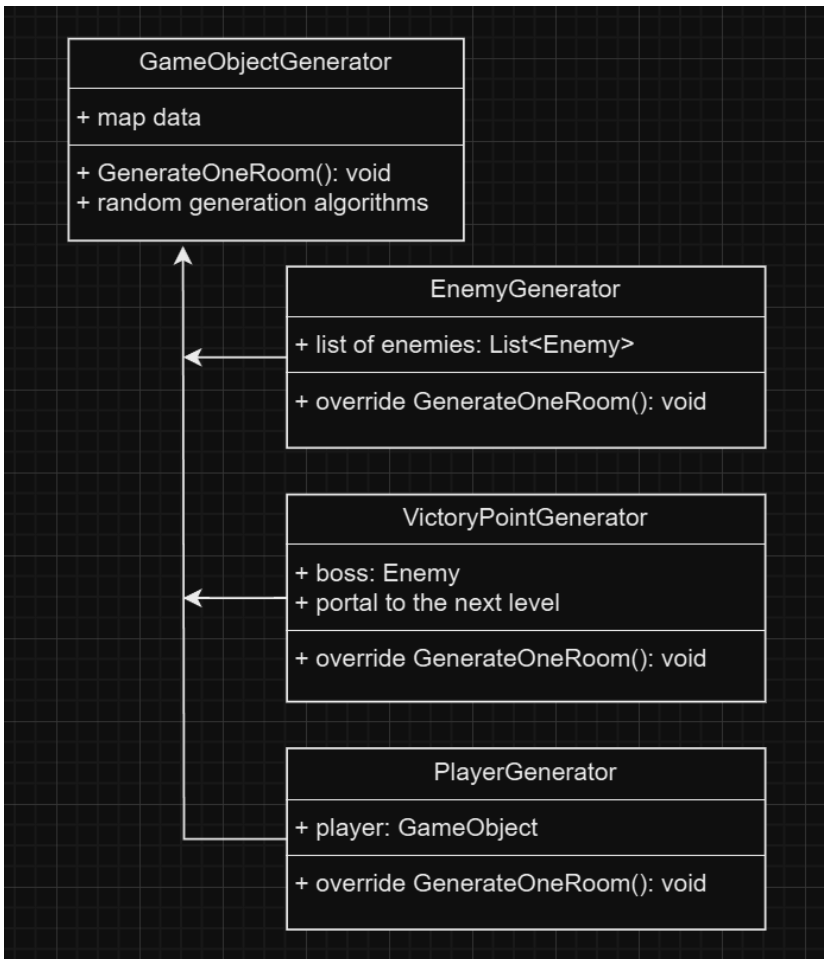
f. GameObjectGenerator

These classes are in charge of randomly generating game objects in a room (the entire map consists of several rooms, at least 3). There are three child classes, EnemyGenerator, PlayerGenerator and VictoryPointGenerator.

EnemyGenerator generates objects in an ordinary room, including enemies and their rewards (loot).

PlayerGenerator generates the spawn room, where there will be only the Player but no enemies. It is a safehouse for the player to start the level.

VictoryPointGenerator generates objects in the final room, where there will be boss enemies and the portal to the next level.



GameObjectGenerator Classes

Design Patterns and Principles

1. Observer Pattern:
   Parameters such as a game character's health bar will only change if it is under attack. It is unnecessary to keep checking and updating the health bar in every Update(). Thus, health bars are registered by attack related interfaces and updated only if hostile attacks are made.
2. Separation of Concerns Principle:
   A different module addresses a separate concern. For example, when coding for Enemy, active behaviours and passive behaviours are addressed by separate modules (Enemy and EnemyHealth).
3. Open-closed Principle:
   As we develop our code, we make minimum modifications to what is already there. For example, for indestructible enemies, instead of relocating the health component from the parent class to every destructible child class, we will just set this component inactive in indestructible enemy classes.
4. Coding Standards:
   a. Function names should start with capital letters, e.g. void TakeDamage(float attack).
   b. Class member variables should start with lowercase letters, e.g. float attackInterval.
   c. Const variables should be named in all caps, e.g. SCENE_NAME.

Design Decisions

1. We have changed our game layout from vertical layout (multiple storeys in the vertical direction, looking from the side) to horizontal layout (multiple rooms on one storey, looking from the top). This is because the latter better presents the interior layout of a castle. Besides, gravity need not be

considered in a horizontal layout, which will make our movement designs simpler.

2. For procedural generation, we have chosen the room-first-then-corridor method over the corridor-first-then-room method to easily and clearly define each room in the map. This is because the rooms are only connected together after generating the corridors.
   The advantage of the alternative method is that it can easily find out a start point/room and an end point/room for the map, because the positions of the rooms are generated based on the route of the corridor.
3. For now, we edited parts of the free sprites for the Player. This allows for more customised animations, which is essential for our weapon system in which the Player can swap to and attack with a list of weapons.
   An alternative is to use free sprites online, which will save us much time. We have been using them for enemies and interactables, because these objects require less customisation.

Testing

Unit Testing

No	Test case	Result
1	Click the 'New Game' button on the Main Menu	The Main Menu is closed, and a new game scene is opened.
2	Click the 'Settings' button on the Main Menu.	The Main Menu is closed, and the settings menu is open.
3	Drag the 'BGM vol' slider in the settings menu.	BGM volume correctly changes with the position of the slider.
4	Click the 'Exit' button on the Main Menu.	The program is closed.
5	During the game, click the 'Pause' button.	The game is paused. The 'Pause' button changed to the 'Resume' button. The 'Settings' button appears.
6	During the game, click the 'Quit Level' button.	The game scene is immediately closed and the Main Menu is opened.
7	After pausing the game, click the 'Resume' button.	The game resumes, the 'Resume' button changes to the 'Pause' button, and the 'Settings' disappears.
8	After pausing the game, click the 'Settings' button.	A settings window appears.
9	From test 9, drag the slider 'BGM vol'.	BGM volume changes accordingly.
10	From test 10, click the 'Back' button.	Settings window disappears. The changed BGM volume applies in game.
11	In the game, press the keys (individually) 'w' 'a' 's' 'd'.	The player character moves in up, left, down, right directions respectively.

12	In the game, left-click the mouse on the left side of the screen.	The player makes an attack to his left.
13	In the game, left-click the mouse on the right side of the screen.	The player makes an attack to his right.
14	In the game, right-click the mouse on the screen.	The player makes a ranged attack, aiming at the cursor.
15	In the game, spam left-clicks.	The player attacks continuously, but enters the next attack after the current attack is finished.
16	In the game, spam right-clicks.	The player attacks continuously, but enters the next attack after the current attack is finished.
17	In the game, an enemy attacks the player, where the enemy's attack is 10, the player's defense is 30.	The player's health drops by 7, which is correct (30% damage reduction).
18	In the game, an enemy attacks the player, where the enemy's attack is 10, the player's defense is 0.	The player's health drops by 10, which is correct (0% damage reduction).
19	In the game, move the player to go against the walls.	Player occasionally goes through the wall.
20	In the game, move the player inside the portal, and press key 'f', without collecting all keys.	Nothing happens.
21	In the game, move the player inside the portal, and press key 'f', with all keys collected.	Game scene is closed. The Transition scene is opened.
22	In the game inspector, set an enemy's health to 0.	The enemy dies (animation) and disappears after 2 seconds.
23	In the transition scene, click the 'next level' button.	The Transition Scene is closed, and a different new level is opened.
24	In the game inspector, set the player's health to 0.	The game transits to the End Scene, correctly showing the player's statistics.

Integration Testing

1	After changing the 'BGM vol' setting, close and open the game again.	The changed BGM volume is applied after re-entering the game.
2	In the first game, move the player near an enemy and left-click.	The player hits the enemy and deals damage, and the enemy's health bar decreases.
3	Continue to attack an enemy until	The enemy dies, and its body disappears. A few items

	its health bar drops to 0.	spawn upon the enemy's death. Kill count record increases by 1.
4	Move the player near an enemy bandit.	The bandit starts moving towards the player and attacks when they are close enough.
5	In the game inspector, set all enemies' healths to 0.	All enemies die. Kill count record increases. Many items are dropped. Look through the hierarchy, the number of keys dropped is equal to the number of keys required to go to the next level.
6	Move the player over the buff items dropped.	The buff items disappear upon contact. In the inspector, the player's statistics have been increased by the buffs.
7	Move the player to collect all the keys around the map.	Upon collection of a key, a key symbol at the bottom left corner of the screen lights up. All symbols light up after the last key is collected.
8	Move the player near the portal and press the key "F".	The game transits to a transition scene, where the player's statistics are correctly displayed.
9	Go to the next level, and let the player be killed by enemies.	In the next level, the level record has increased by 1. After the player is dead, the game transits to a game-over scene, where a table of records and statistics is displayed.
10	After this game, both the kill count and the level records are broken.	In the game-over scene, the current level and the current kill count are in red (highlighted), while other texts are in white.
11	Go back to the main menu and start a new, second game, and set the player's health to 0 through inspector.	The game transits to a game-over scene. Both the level record and the kill count record have been updated to the record of the first game. All texts are white as there is no record broken.

User-testing

At the game developing stage, we test out in the game mode every time we have completed a new feature. Then we fix any bugs we observed immediately, or log it in GitHub issues. We have followed this procedure for all the features in our game to ensure that there are minimum bugs in our game before we finalise the project.

In addition, after we have completed the major features of the game, we build the project and exported it into exe and let others test the game by playing them. We have managed to gather a lot of feedback on bugs that we have overlooked as well as how we can improve the game.

1. User Feedback 1:
   The player got stuck when encountering the Evil Wizard. While the Wizard kept dealing damage to the player, I could hardly move or attack due to the injury animations. I always died here.

2. User Feedback 2:
   The rewards sometimes drop outside the map. I could not pick them up. Keys seem to have the same problem, but I so far have not gone to an incidence where they completely went out. Part of the keys were still inside the map so I could still pick them. I am not sure what would happen if

I lost some of the keys in a level.

3. User Feedback 3:
   I did not know what keys to use. I guessed movements should be using WASD, which was correct. But it took me a few minutes to find out how to pick up weapons. Also, I only realised the ranged attacks were aiming at the curser. I think a tutorial or at least some hints should be made.

4. User Feedback 4:
   After the BGM ended, it just stopped playing. Maybe you can add more music, or make the BGM loop.

5. User Feedback 5:
   The menus were too simplistic. Just plain buttons and a blue background, which looked very lame. Should make better ones.

6. User Feedback 6:
   The bringer of death boss was quite nicely done. It had ranged attack and melee attack, and the attack intervals were quite balanced.

7. User Feedback 7:
   For the attacks, I only found out in level 3 that if I clicked on the left side of the screen then I would attack left, and right side of the screen to attack right. At first I thought it was a bug, that the attacks sometimes went left and sometimes went right.

8. User Feedback 8:
   The stats in the transition page was quite useful, to see how powerful my character has become, so I could see whether I could play more aggressively in the next level. But the texts were too plain.

9. User Feedback 9:
   Overall the game was quite smooth. I played a few times and it did not crash and I did not find any major bugs that could make the game unplayable. There were small bugs, or imperfections. But I think they did not affect the entire thing.

10. User Feedback 10:
    The ranged attack using the magic book was very cool. It was a surprise that the attack could push back enemies also. Details like this motivated me to explore deeper.

11. User Feedback 11:
    I did not know the effects of the blue bottles and red bottles. I just picked up and I could not see any visible effects. Also I think there could be more rewards. So far I only saw 3 bottles of different colours, and I felt it was not enough.

12. User Feedback 12:
    The menu was simplistic, but quite straight forward. I could see what to click for the first time.

13. User Feedback 13:
    The maps were quite ok. I knew you were using free stuff online, so I understood the arts were not perfectly done. But the variety was there. I played a few rounds, and everytime I got really different maps.

14. User Feedback 14:
    The enemies, especially the bosses, were very cool. They had their unique attacks and movements, and some of them were really challenging. The animations were very well done.

15. User Feedback 15:
    One improvement I want to bring up is that the boss room and teh spawn room were a bit too empty. Maybe you can stuff more things or enemies inside.

Known Bugs

1. Bug:
   Player sprite occasionally penetrates or goes inside the wall tiles.
   Possible cause:
   There might be gaps in the tilemap, or the z positions are different between the character and wall.

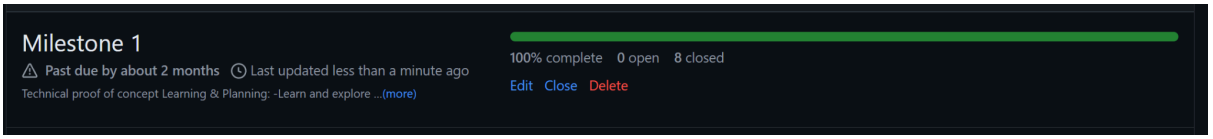
2. Bug:
   Player gets stuck in the enemies' collider component.
   Possible cause:
   Variables used for pathfinding of enemies are not calibrated such that the enemies will not stop even after its collider components coincide with that of the players'.
   Solution:
   We have added a transform child object that is placed outside the enemies collider, so that the enemies will not move into the player object.

3. Bug:
   The key and rewards dropped slip into/outside the wall and cannot be picked up.
   Possible cause:
   The bouncing distance is too far.
   Solution:
   We increased the size of the collider of the keys, so that it can be picked up even if it's inside the walls.

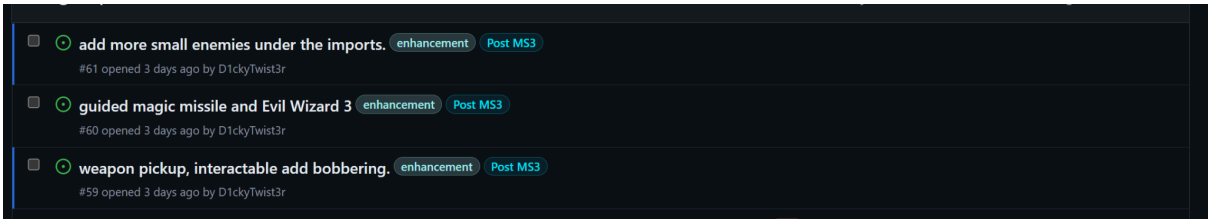
4. Bug:
   One of the maps has problematic walls. Characters can walk over and walk through the walls.
   Possible cause:
   Bad collider settings, or the tile sprites used may not have a proper collider.

Version Control

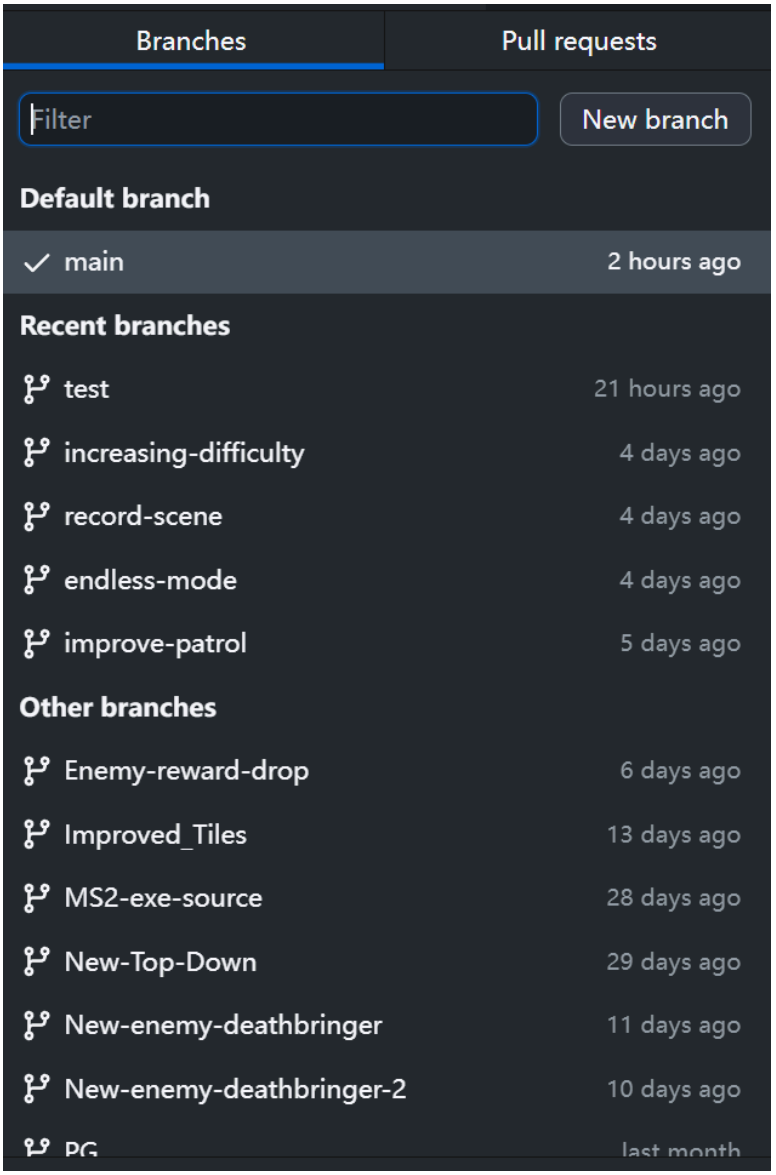
We have been using GitHub for version control in this project. We split our tasks into mainly 3 milestones, and create issues for these tasks and any problems. New branches are created to settle these issues. Pull requests are made and accepted after making sure no major bugs or conflicts occur.



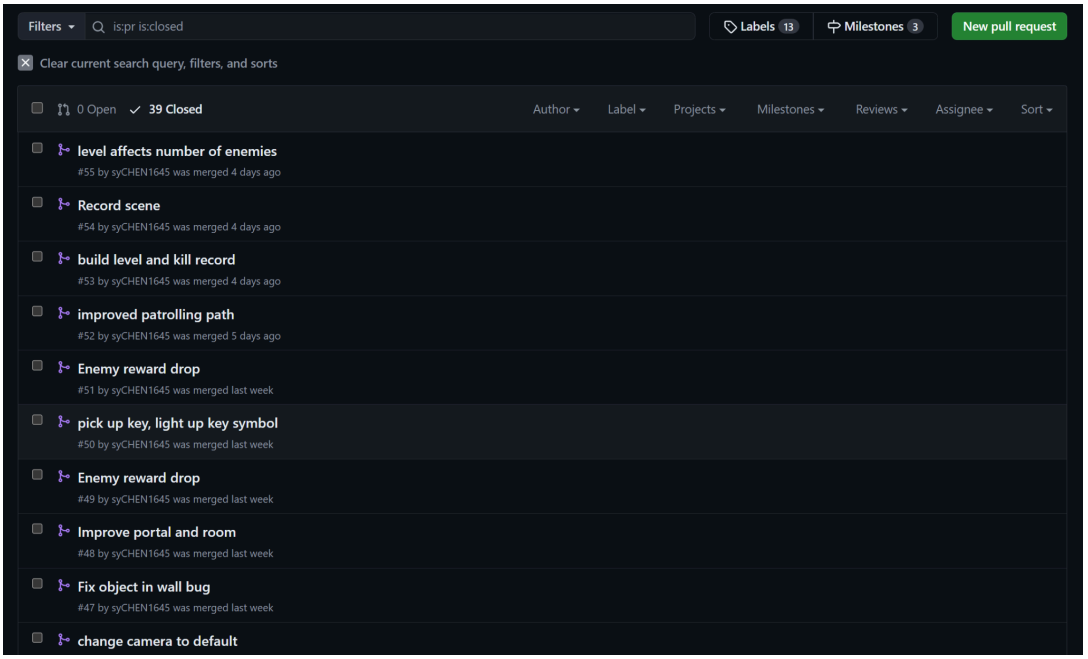
Milestone



Some Issues Created for Bugs Tested



Some of the Branches to Solve Different Issues



Some Pull Requests

Tech Stack

1. Game engine: Unity.
2. Version control: GitHub.
3. Coding platform: Visual Studio Code.