Department of Electrical Engineering

FINAL YEAR PROJECT REPORT

BENGEG4-INFE-2019/20-KC-06

Creation of a computer game

Student Name: Lam Ting Hin
Student ID: 54816737
Supervisor: KC - Prof CHIANG, K S
Assessor: LHC - Dr CHAN, Leanne L H

Bachelor of Engineering (Honours) in
Information Engineering

Student Final Year Project Declaration

I have read the student handbook and I understand the meaning of academic dishonesty, in particular plagiarism and collusion. I declare that the work submitted for the final year project does not involve academic dishonesty. I give permission for my final year project work to be electronically scanned and if found to involve academic dishonesty, I am aware of the consequences as stated in the Student Handbook.

Project Title : Creation of a computer game

Student Name : Lam Ting Hin

Student ID : 54816737

Signature Cody

Date : 15/4/2020

Turnitin Originality Report:

Abstract

Since the 1980s, video games have become an increasingly important part of the entertainment industry. Nowadays, video games already become a part of people's life.

This project aims to create a computer game from scratch. The game is a 3D role playing game focus on killing enemy and exploring map. In the game, player is a knight who is trapped in a dungeon which is full of monsters and traps. Player have to explore the dungeon and find a way out. The goal of the game is to kill the final boss (The Gatekeeper) and escape from the dungeon.

The game was created in Unity, the most popular and user-friendly game engine. The game map was designed from scratch and built in Blender.

Many of works have been done to create fun game play. The game is still a prototype, it still need further improvement, but the game mechanism that I'm aiming is already made. This is the game I want to make.

Acknowledgements

I want to thanks my supervisor Professor Chiang, Kin-Seng for giving me this opportunity to develop a game as a school project. Although we are not able to meet in school because of coronavirus, Prof Chiang and I have set up meetings via Zoom and he gave me a lot of useful feedbacks for improving my game, the game is much better than it was thanks to Professor Chiang helps.

Also, I want to thank my friends for becoming the test player of my game, their feedback are also useful, my game will not be finished without their help.

Contents

List of Figures

List of Tables

1. Introduction

The objective of the project is to create a computer game with any open application software available or software developed by myself. My creativity and artistic talent will be demonstrated in this project.

1.1. Background

The computer games should fulfill following requirements:
The application should be able to run on medium-end computers with suitable operating system. The game should be playable. A goal should be set up for player to achieve in the game, and it have to be achievable. The game should be fun. Player play game for entertainment so the game must not be boring. The game should be creative. There should be something new in the game that is different from those games in the market, that makes player having fresh feeling.

According to “10 Things Great Games Have in Common” (gamedesigning.org), a great game should have these 10 elements:

1. Great Control: good control that player can interact with the character easily.
2. Interesting Theme & Visual Style: good visuals
3. Excellent Sound & Music: comfortable/suitable BGM
4. Captivating Worlds: interactive world that can be explored
5. Fun Gameplay: interesting main game mechanics, most important.
6. Solid Level Design: progressively getting more challenging while remaining fun to play.
7. Memorable Characters: interesting character that player can relate to.
8. Good Balance of Challenge & Reward: good escalation of difficulty.
9. An Entertaining Story: serving as a storytelling medium that no other can compare to.
10. Something Different: offers a new experience to player.

1.2. Objectives

- Study gaming background for further development
- Methodology and work approach
- Game Structure Design
- Learning about writing script to animate characters.
- Learning about 3D modelling & Animation for creating my own assets.
- Create player avatar, enemy and map
- Create Animation and Animation controller
- Design Game UI
- Add Graphic effect
- Add Music
- Debug & Balancing

2. Methodology

2.1. Choosing Game Engine

I decide to use Unity as the game engine to develop the game because Unity have a lot of benefits compare to other game engine

Free for personal use:

Unity Personal is a free version of Unity for individuals, hobbyists, and small organizations with less than $100K of revenue or funds raised in the last 12 months. I am eligible to use Unity for creating game for the Final Year Project, with no fee.

Unity Assets store:

Unity provide a platform for users to share their assets. There are massive amount of assets uploaded in the store, and some of them are free. Modelling takes time, I need to spend a lot of time on create a beautiful character/map asset by myself. The FYP is an individual project so I may not have enough time to makes everything looks perfect. The assets store gives me an option to borrow some of the model from the others so that I can save time on modelling everything by myself and focus on the game design.

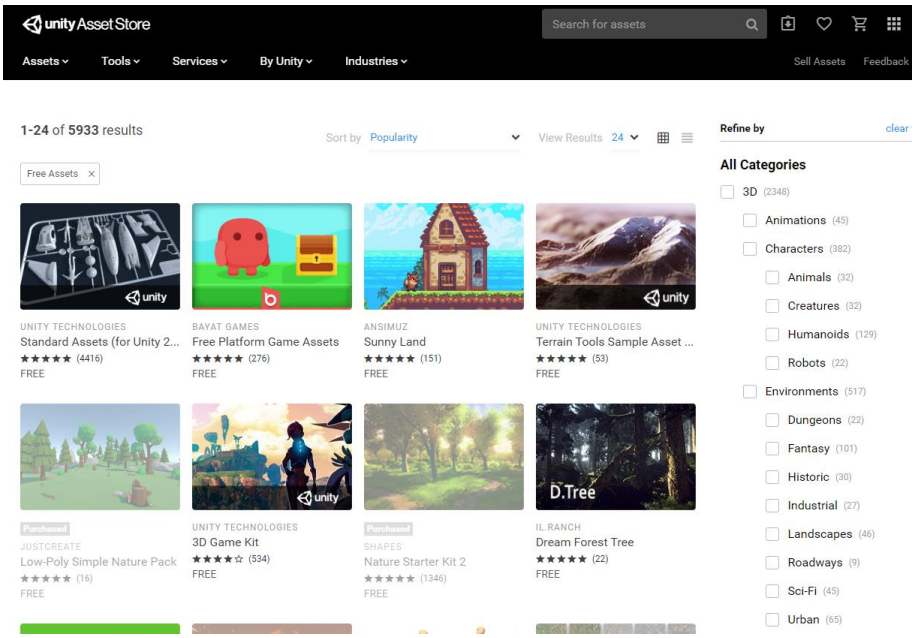

Figure 1 Unity Assets Store

Tutorial Project:

Unity also provide a lot of tutorial project for beginner to get familiar with it. Those tutorials provide detail instruction to guide me through the game making process step by step.

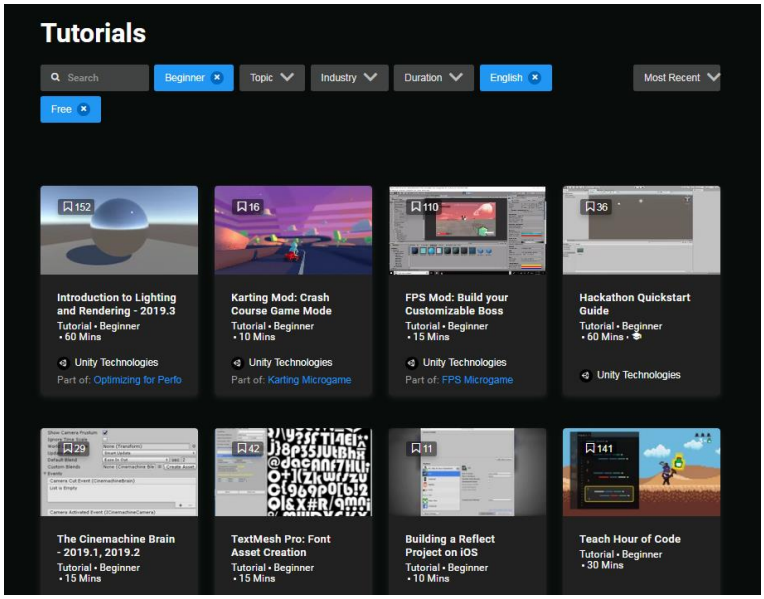

Figure 2 Unity Learn

Huge Community:

Unity is the most popular game developing tool in the world, the community of developers is huge. Therefore, I could easily find the solution on the internet for most of the problem I will faced. I'm a beginner on game developing, so I will properly face a lot of problem, it will be good if I can find the solution easily.

2.2. Unity

Unity is a cross-platform game engine developed by Unity Technologies. Unity gives users the ability to create games and experiences in both 2D and 3D, and the engine offers a primary scripting API in C#, for both the Unity editor in the form of plugins, and games themselves, as well as drag and drop functionality.

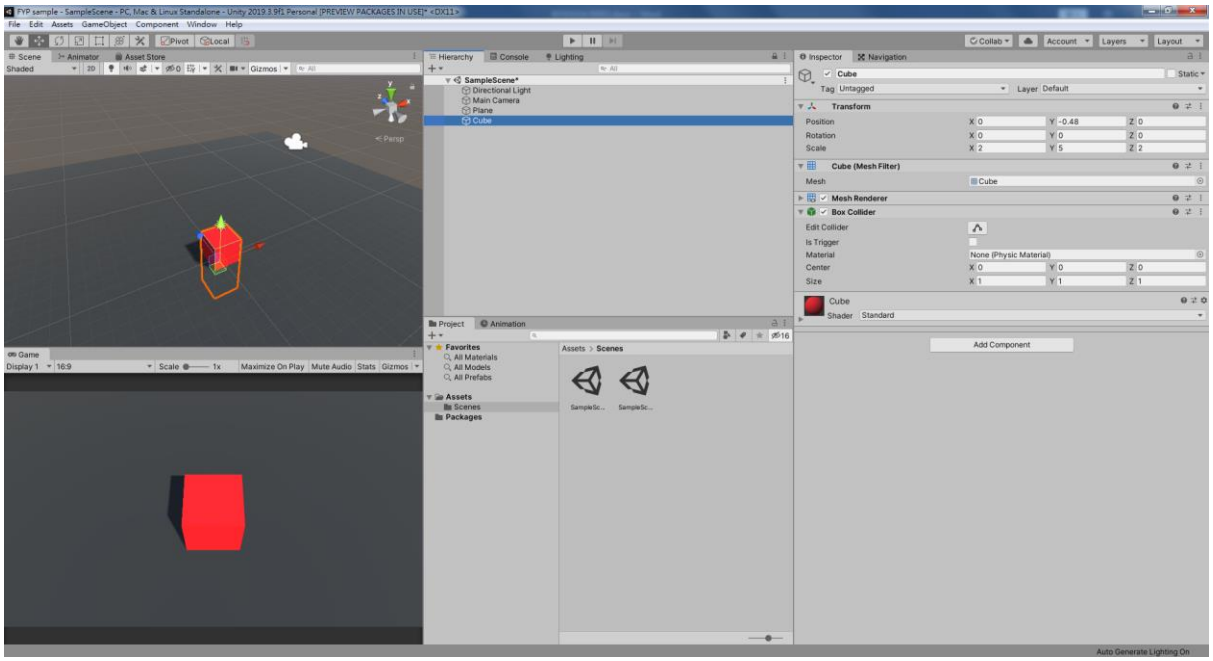

Figure 3 Unity

To learn about how to use Unity, I strongly suggest student to watch this tutorial if you know Japanese, it is a well-made tutorial that teach you how to make a basic game step by step. It is a free course so no worry about money.

ユニティちゃんが教える！初心者向け Unity 講座

Miss Unity teach! Unity Lecture for beginners

https://www.udemy.com/course/unity-chan-tutorial-01/

Figure 4 Unity Tutorial Course

2.4 Blender

Blender is a free and open-source 3D computer graphics software toolset used for creating 3D printed models. Blender's features include 3D modeling, UV unwrapping, texturing, etc. In this project, Blender was used to create the model of the gaming map.

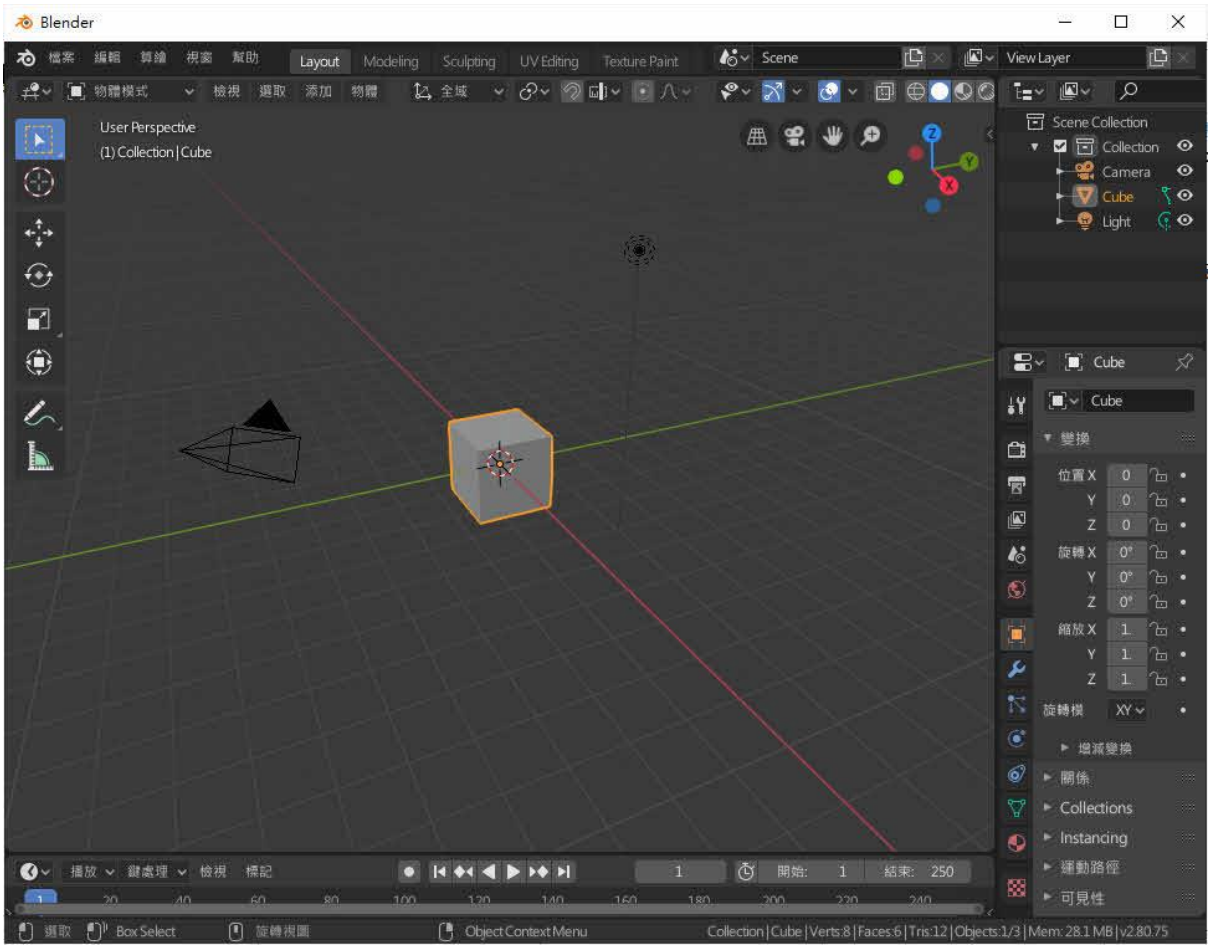

Figure 5 Blender

3. Design

3.1. Game Genre

To decide the type of game is the first thing we need to do when we develop a game, you need to know what you want to make.

There are a lot of game genre in the world, shooting game, horror game, racing game, sports game, etc. All of them are fun, but I have to make a choice.

I decide to make a 3D Action Role-Playing Game (RPG), which player is going to role-play and control a character to explore the gaming world. I choose this genre because my favourite game series “The Legend of Zelda” is also 3D action RPGs.

Player role-play the main character “Link”, explore dungeons, kill monsters, and save the world at the end. The game series has been a huge success worldwide and I really love it. Therefore, I decide to create a game like this.

https://metro.co.uk/2011/06/16/the-legend-of-zelda-ocarina-of-time-3d-review-gaming-royalty-46370/

Figure 6 The Legend of Zelda

3.2. Game Story – Objective of the game

It is not necessary for a game to have a big lore, but at least it need some plots to tell players about the goal of game, otherwise players will not know what to do.

In my game, player will role-play a knight who is trapped in a dungeon which is full of monsters and traps. Player have to explore the dungeon and find a way out. The objective of the game is to kill the final boss (The Gatekeeper) and escape from the dungeon.

3.3. First Person vs. Third Person Perspective

First-Person Perspective:

In video games, first person is any graphical perspective rendered from the viewpoint of the player's character, or a viewpoint from the cockpit or front seat of a vehicle driven by the character.

Third-Person Perspective:

Also known as the "over-the-shoulder" view. A third-person perspective is when the player can see the character he or she is playing onscreen, more of the environment can be seen while playing

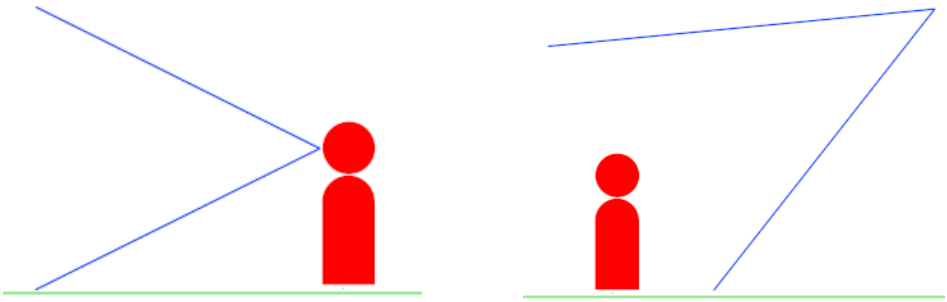

Figure 7 Camera Perspective

Comparing two of them, third-person view have wider angle of vision, since my game is a dungeon exploring game, I wish players will watch and enjoy the dungeon I made, so it is good for players have wide field of view, so I choose third person perspective

4. Implementation

4.1. Game Environment

4.1.1. Map

Map model was designed from scratch and built in Blender. The map made out of several rooms, and rooms made out of grids Using grid to create rooms is an easy method on designing map, it is easy to state the size on the design diagram using grid size and you can easily predict the size of the whole map. A normal room made out of 3x3 grids, and some special rooms have different size. For example, the boss room in Stage1 is a 5x5 room because I put 3 enemies in that room and it needs more space.

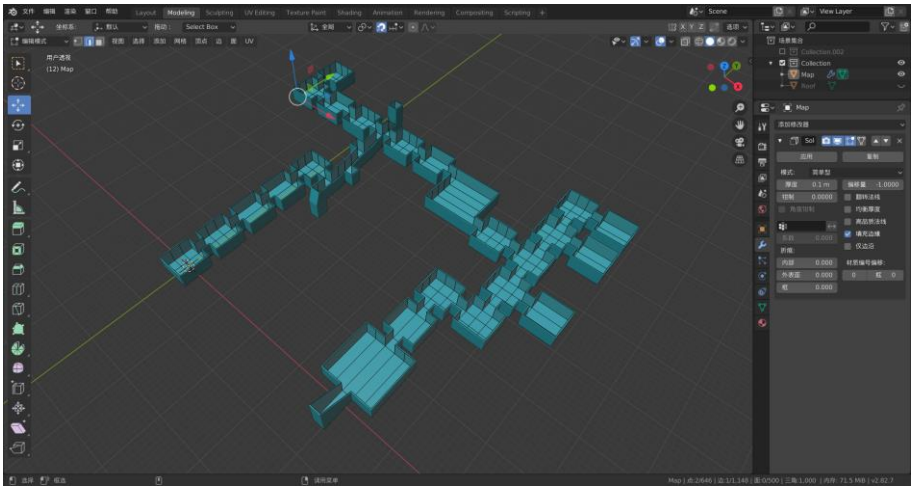

Figure 8: Game Map – Stage 1

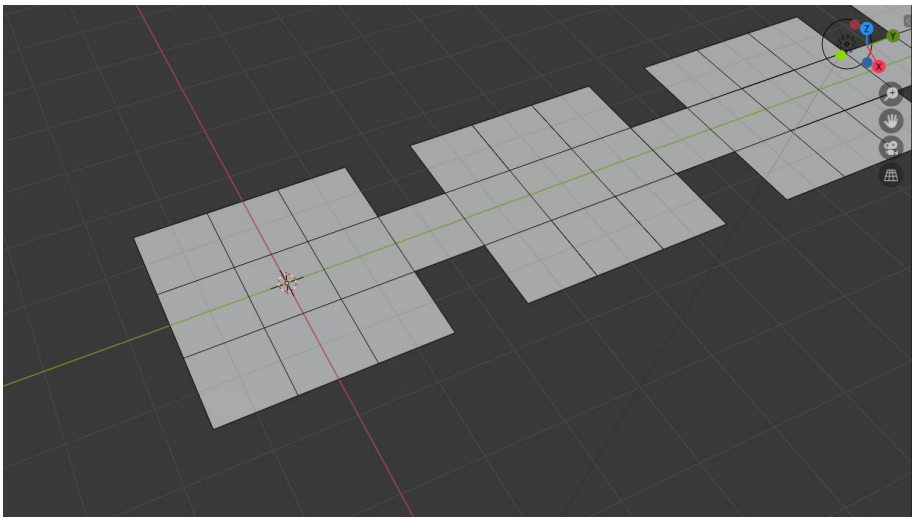

Figure 9: Game Map – Room

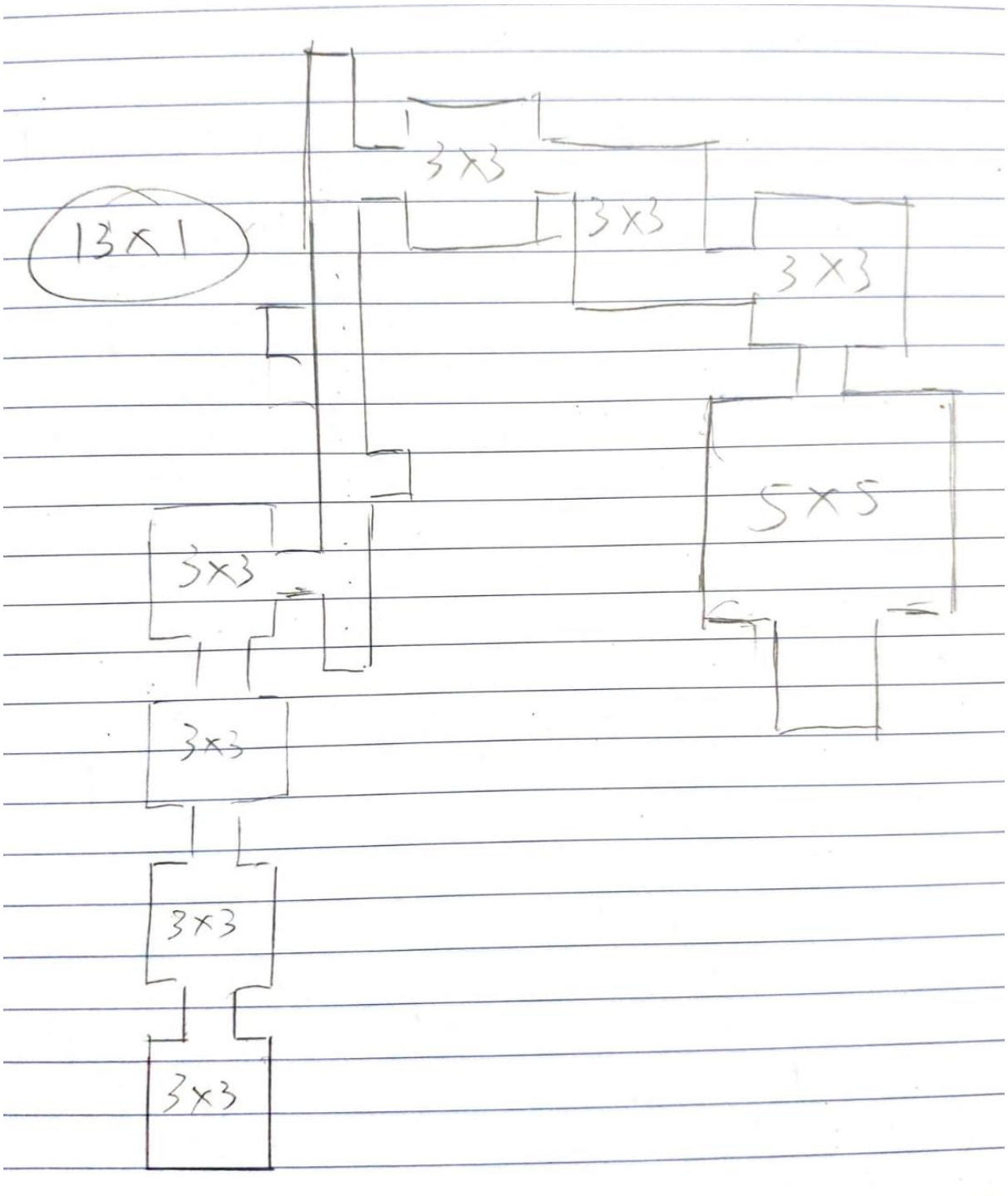

Figure: Game Map – Design First Draft

4.1.2. Lights

We need lights to lighten up the map environment, the map will be dark without lights.

I used the “Modular Medieval Lanterns” assets from the assets store and add light component to the model to create a light game object. I duplicate the game object and put them into every rooms to lighten up the whole map.

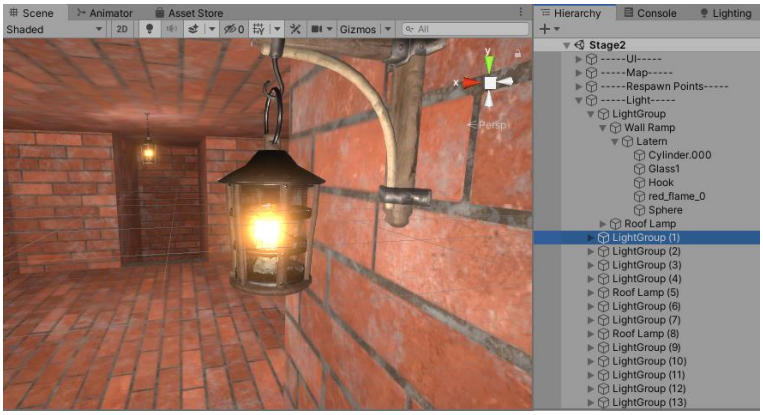

Figure 10 Light - Lantern Model

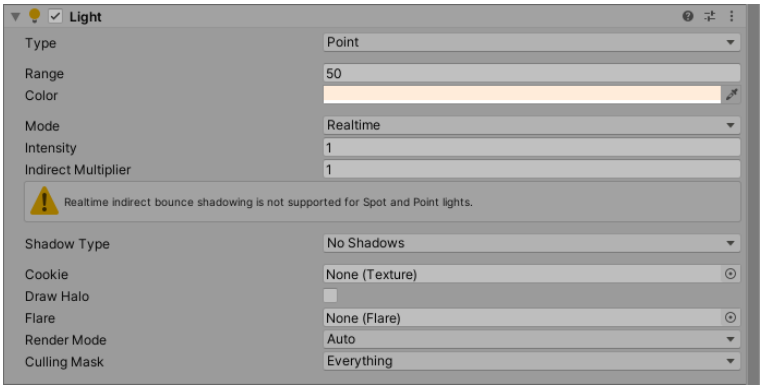

Figure 11 Light - Light Source Component

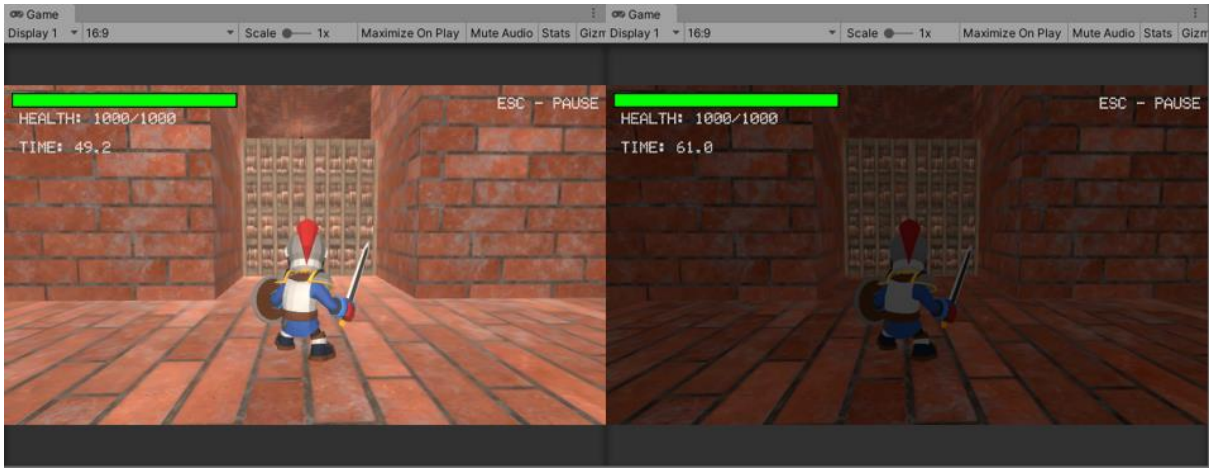

Figure 12 Light - Comparation

4.1.3. Gate

Although the main objective of the game is to escape from the dungeon, the main game play of the game is to: 1. Explore the map 2. Fight monsters. The game will not be challenging and be lacking excitement if player is able to walk directly to the goal. Therefore, I place some gates in the map to block the road to the goal. There are two type of gates:

Enemy Gate:

This type of gate will open only if all the enemies in the room are dead.

Figure 13 Gate - Enemy Gate

Enemy objects were assigned to the gate scripts to form an array of game object. When enemy die, their game object will be destroyed and the array slot will become Null. If all slot in the array are Null then the gate will open. This force player to fight monsters in order to open the gate.

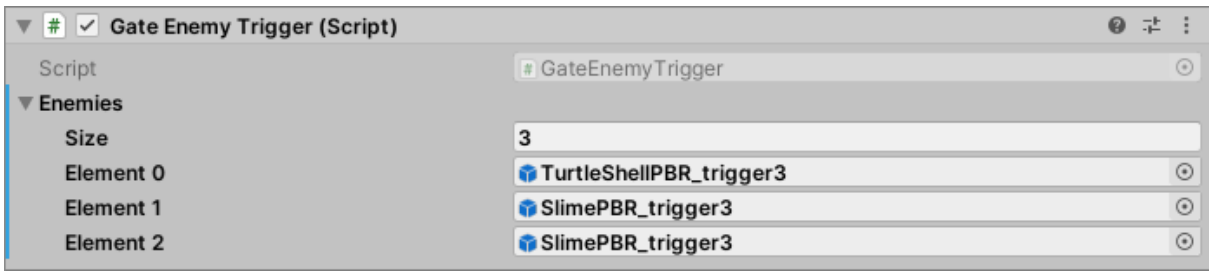

Figure 14 Gate – Enemy Trigger

Key Gate:

This type of gate will open when you turn on the switch of that gate.

Figure 15 Gate - Key Gate

This type of gate will open when you turn on the switch of that gate. Collider trigger was attached to the switch and it will continuously check player's keyboard input while player stay inside the trigger. Gate will open when player press the right key. The switch is placed near to the gate in the figure but some of them a placed far away from each other. Player needs to explore the map to find the switch to open the gate to next room.

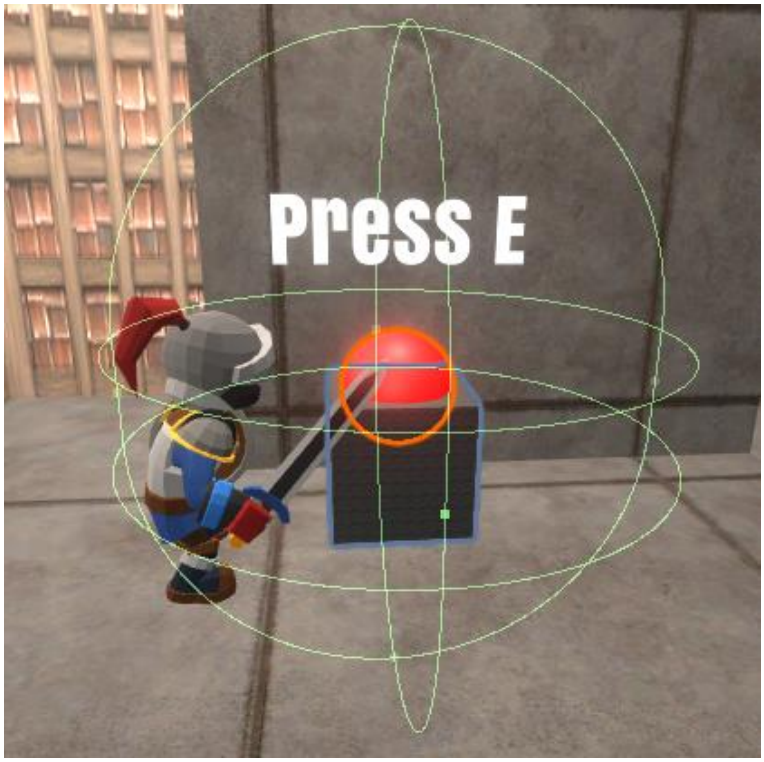

Figure 16 Gate - Key Trigger

4.1.4 Trap

I created traps and put them into the map in order to make the game more challenging. Colliders were attached to these traps and they will deal damage to player if player get hit. Player not only need to fight monsters but also have to dodge traps.

Trap Name	**Damage**
Rolling Rock	200
Laser Pillar	100
Spinning Cutter	150
Swinging Axe	200
Falling Platform	N/A (falling damage)

Table 1 Trap Damage

Rolling Rock: Rocks keeps rolling down from the slope, player need to walk into the hole on the wall to avoid getting hit by the stone and walk forward until next rock comes.

Figure 17 Trap - Rolling Stone

Laser Pillar: A slow spinning pillar which shoots lasers, player will hit the laser if they move too fast, player need to walk slowly to avoid getting hit.

Figure 18: Trap - Laser Pillar

Spinning Cutter: A quick spinning cutter, player need to jump to avoid getting hit.

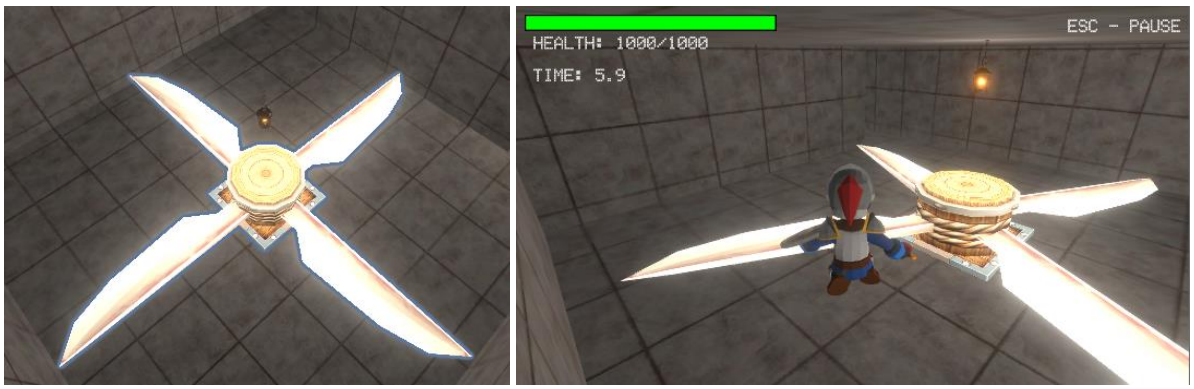

Figure 19: Trap - Spinning Cutter

Swinging Axe: Swing Axe which will knockdown the player, player will fall from the narrow if they get hit by the axe. Player need to walk through the narrow in the right timing to avoid getting hit.

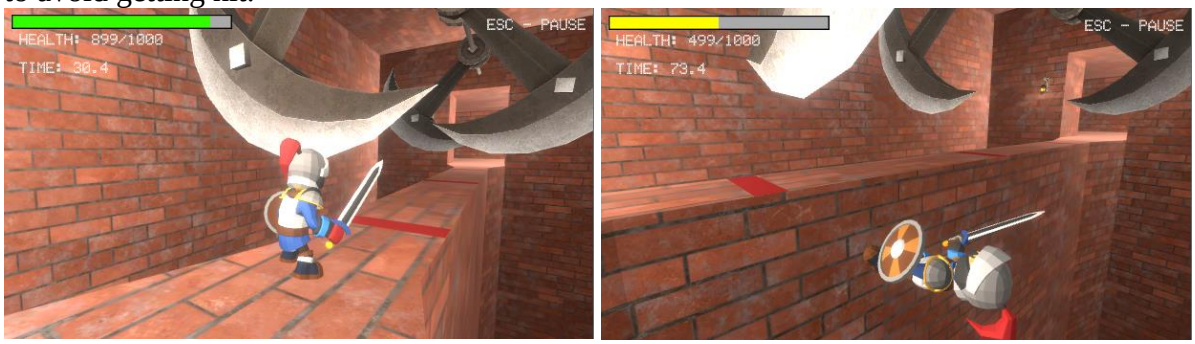

Figure 20: Trap - Swinging Axe

Falling Platform: A platform that will drop if player standing on it, player need to jump to the next platform quickly in order to avoid falling.

Figure 21: Trap - Falling Platform

4.2 Player

4.2.1. Assets

Player character model is one of the most important elements in a game. The character will represent player in the game. It has to be good looking to attract people to play the game. Player model is downloaded from assets store. It is made by Dungeon Mason. This person creates high quality assets and some of them are free. The player model is very well made and have good graphic design.

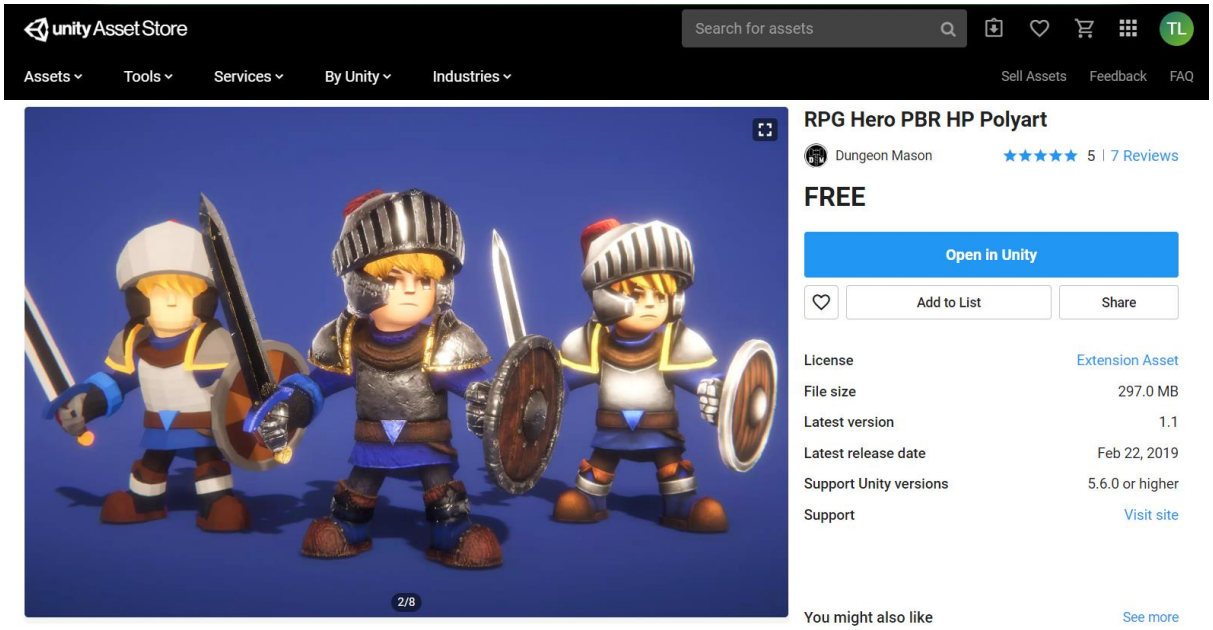

Figure 22 Player - Assets

Besides, the asset also contains animation clip for the model. Therefore, I can easily animate the character by combining these animation clip in the animator.

Figure 23 Player - Animation Clip

4.2.2. State

Player state is a script written for handling players state such as maximum health, current health, etc. Player current health will be set to maximum when game starts and it will reduce when player get hit by enemy or trap or falling from edge, when health reduced to 0, game over. Player health will regenerate in a certain rate, now it is set to 10 in the figure, that means player will regenerate 10 health point per second. Player will respawn at the nearest respawn point when they fall from the edge of the map and its help will be reduced.
When player take damage, it will enter invincible mode for a few seconds, player ignore any damage in this mode. Player will be invincible for 1 second when they get hit and be invincible for 2.5 second when they knocked down.

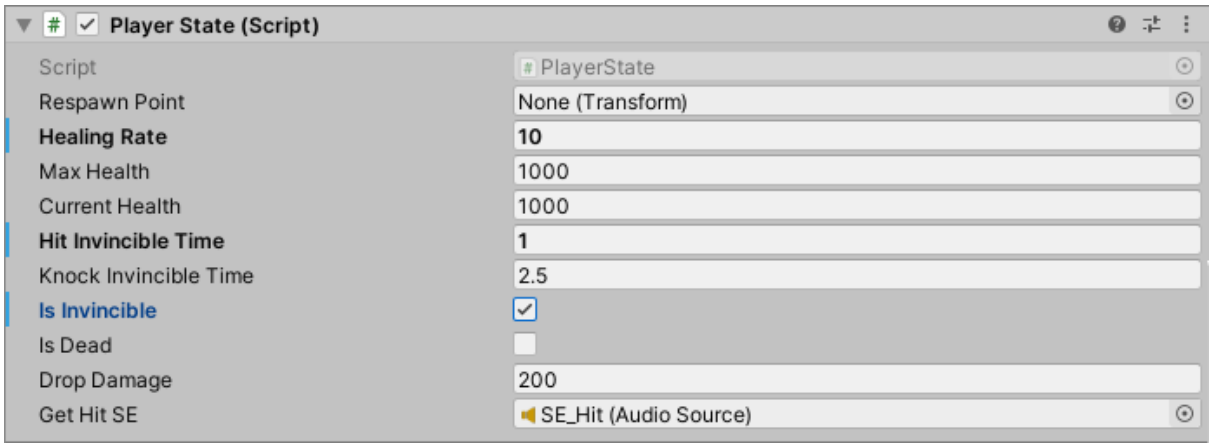

Figure 24 Player State

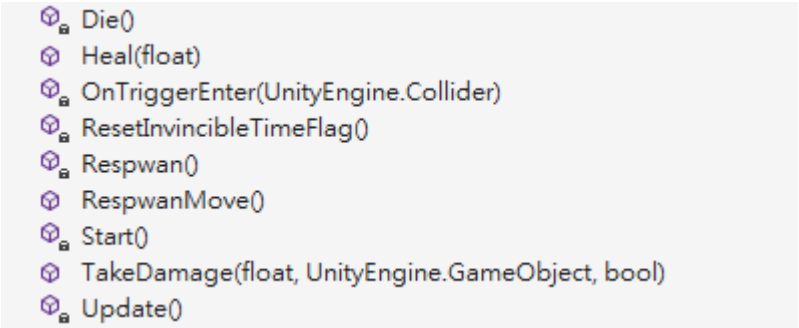

Figure 25 Player State Class View (Method)

4.2.3. Controller

Player controller is written to handle character movement and animation. It keeps checking player's input every frame. Player controller collect the input axis of horizontal and vertical, normalizing the vector, then move character toward input direction. It also helps player to perform jump and attack action. Some public method will be called by player state script to play animation when player take damage.

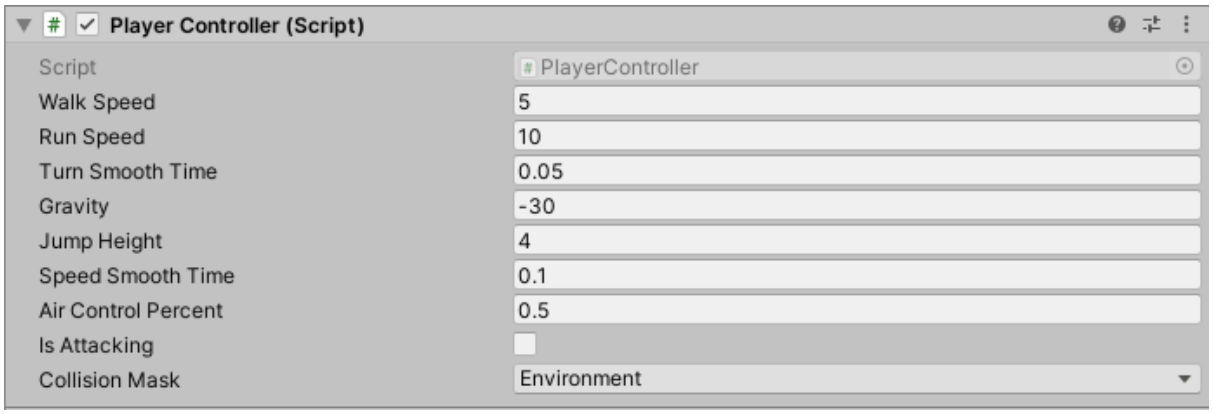

Figure 26 Player Controller

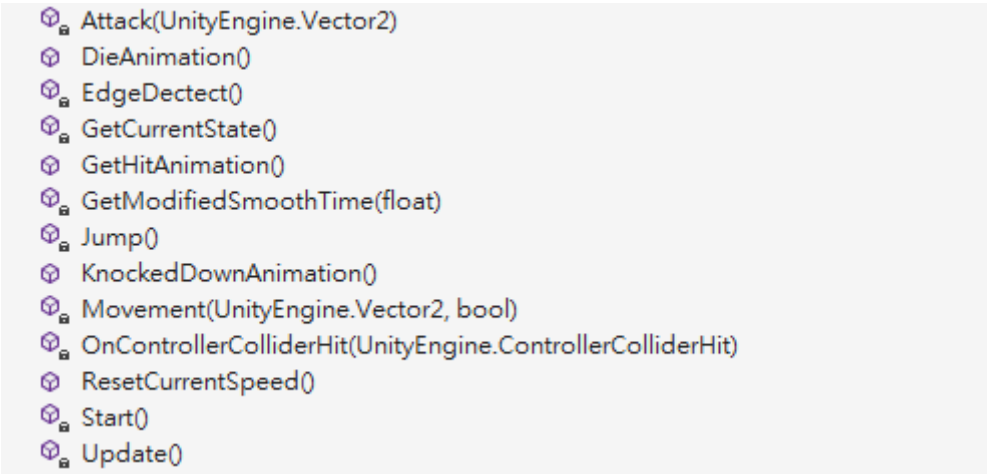

Figure 27 Player Controller Class View (Method)

4.2.4. Animator & Animation

Some animation clips come with the character model so I don't need to create them by myself but I still need to design an animator to handle the animation.

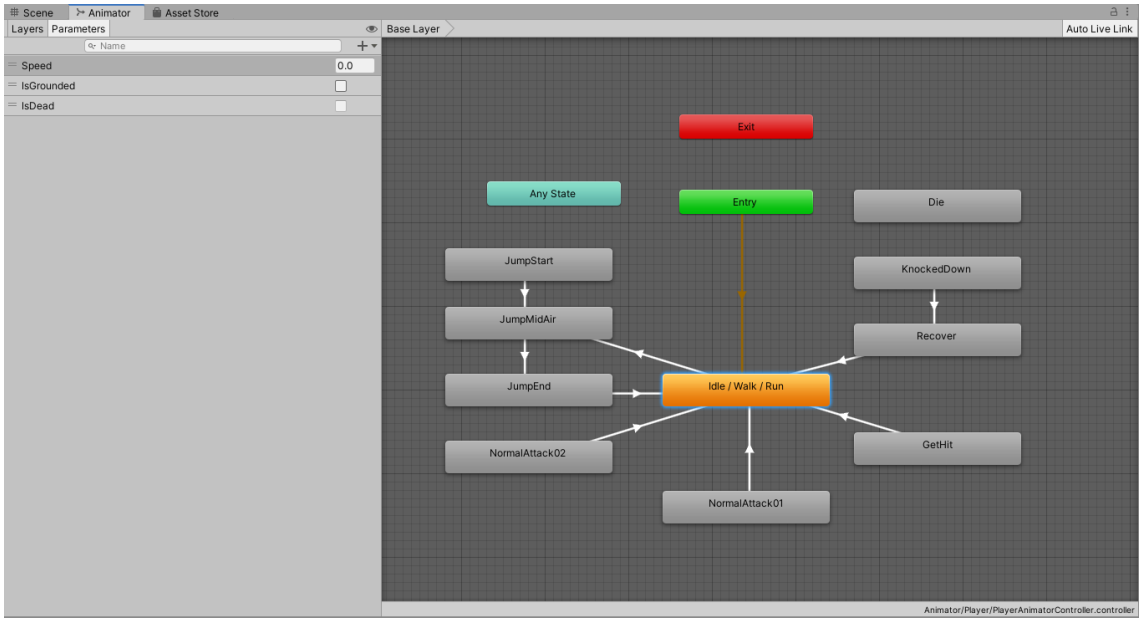

Figure 28 Player – Animator

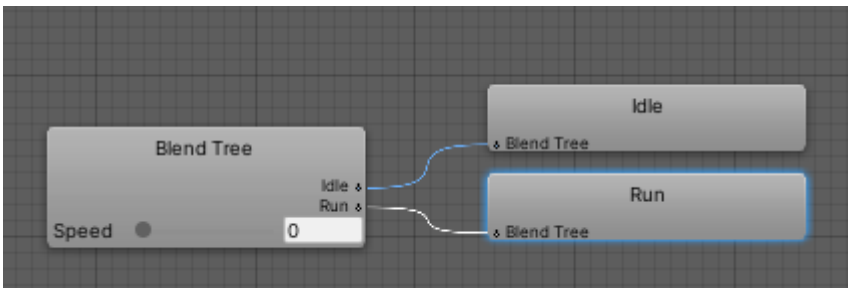

Figure 29 Player – Animator Normal State

Animator is a component that determine when the clip to play, which clip to play.

For example, when player perform jump action, **JumpStart** clip will be played, then it will transfer to **JumpMidAir** when **JumpStart** finished. **JumpEnd** will be played when player hit the ground then it will back to the normal state **(Idle/Walk/Run)**.

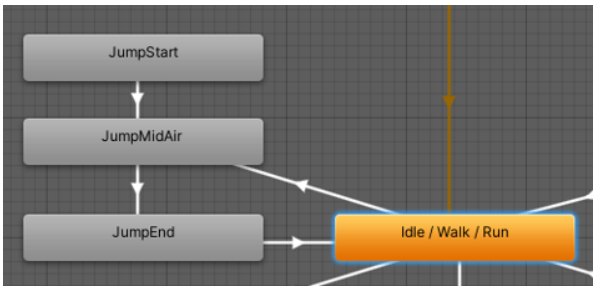

Figure 30: Player - Jump Animation Flow Chart

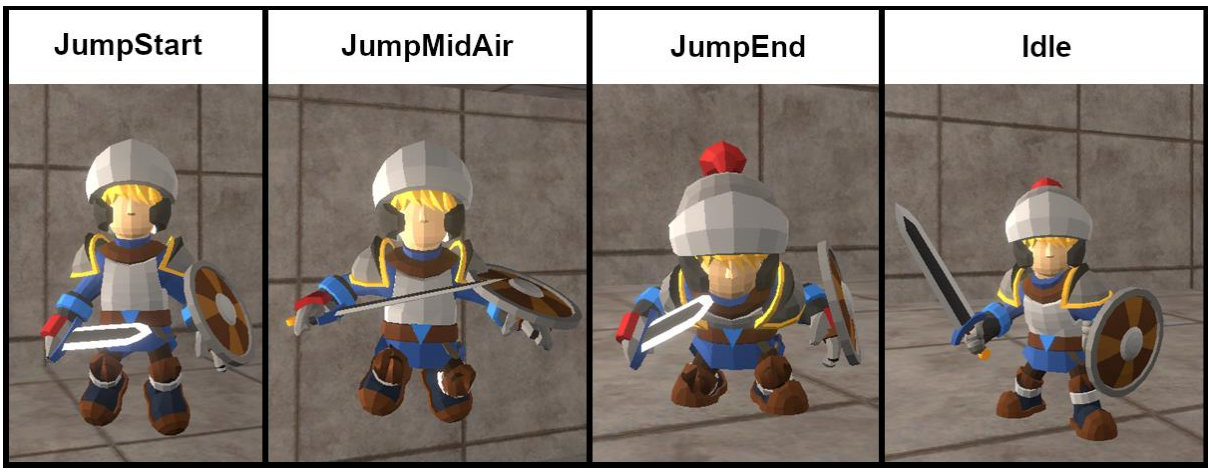

Figure 31 Player - Jump Animation

4.2.5. Hitbox

There is a collider trigger attached to player's sword, it will cause damage to enemy when it hit the enemy's collider. However, it should not be able to hit the enemy when player is not attacking. I want it to be only able to deal damage to enemy during the attacking animation.

Figure 32 Player Hitbox – Sword Collider

Therefore, I edited the animation clip and added new property to it. I added the sword collider property to the attack animation clip so that the collider will only be enabled in certain frame in the attacking animation.

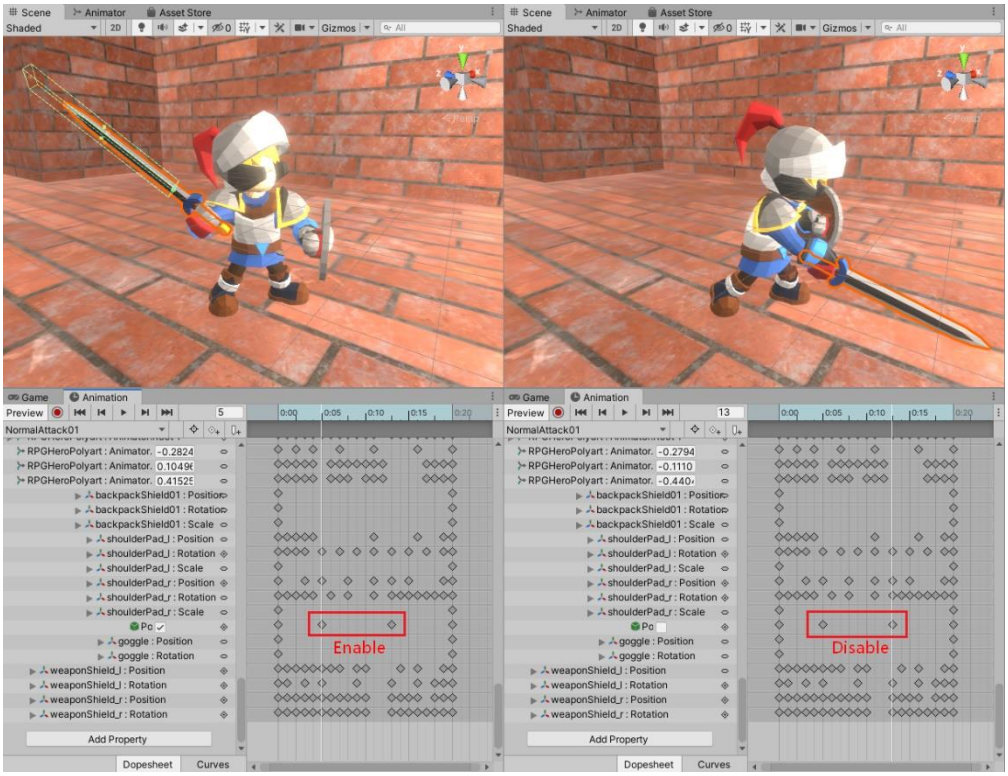

Figure 33 Player Hitbox – Animation Clip

4.2.6. Edge Detection:

The “Character Controller” component provide basic control to player character. However, in some case the result is unnatural. As we can see in the figure, when player standing on the edge, it will not fall from the edge but standing mid-air.

If we look into the scene view, we can see some part of the collider is still touching the ground. To solve this problem and make the fall-from-edge motion natural, an edge detection feature has been made.

Figure 34 Stand on Edge – Game View

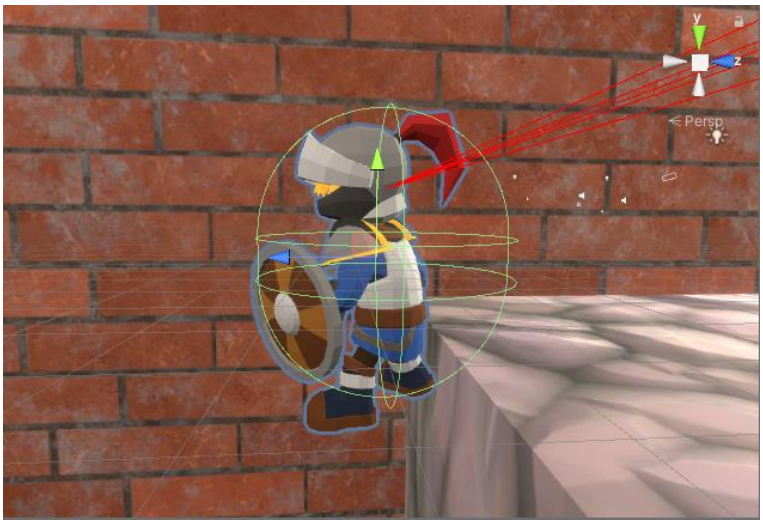

Figure 35 Stand on Edge – Scene View

Rays were cast from the origin of player character to different direction, it returns signal if it hit the ground. As shown in the figure on next page, when player standing on a plane, all rays are green, but if player stand close to the edge, some of the rays turn red. These rays are able to tell the player controller about the edge direction, so that it knows which direction to fall when player is not touching the ground.

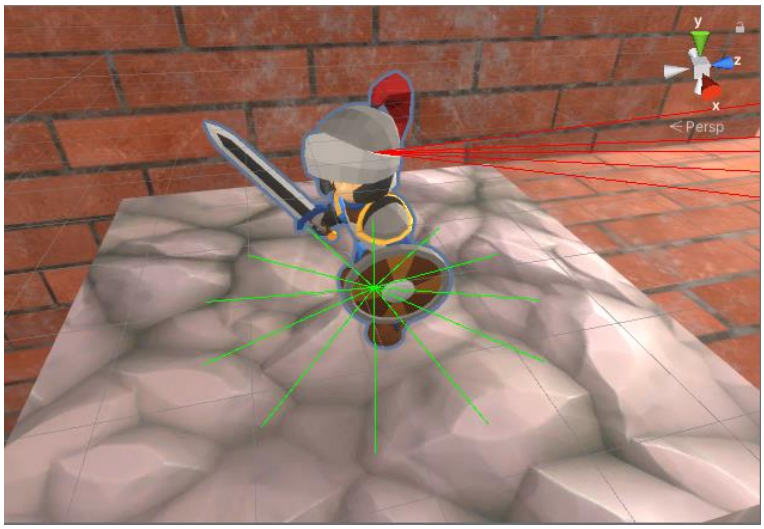

Figure 36 Edge detection – Ray cast hit

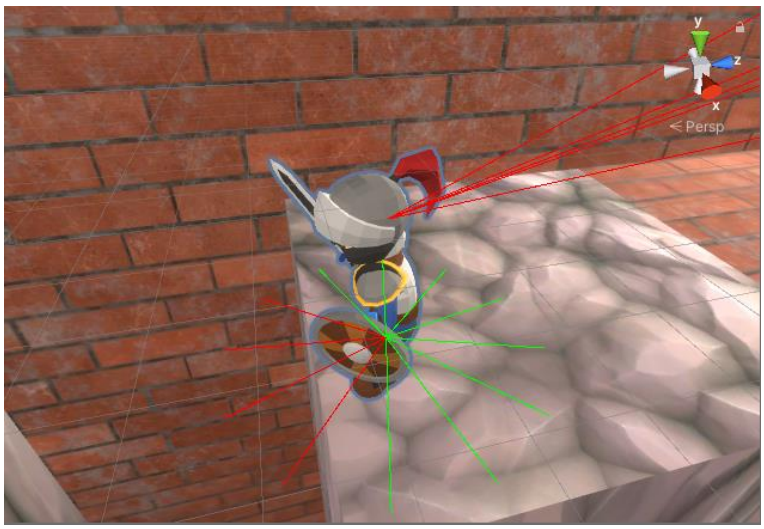

Figure 37 Edge detection – Ray cast miss

A ray is also casted from player downward, it will trigger the droping function when the ray miss, then player character will be forced to more toward the edge and will immediately fall from the edge.

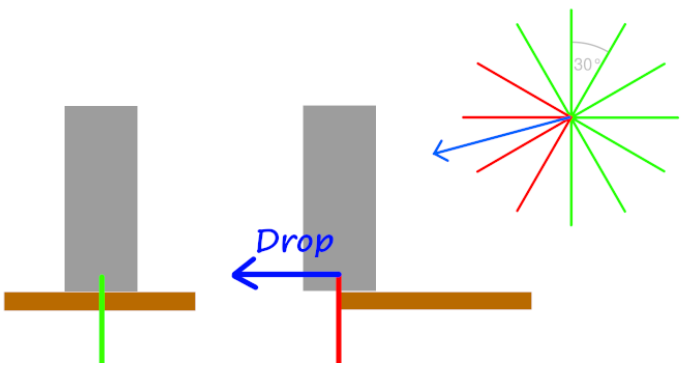

Figure 38 Edge detection – Move Direction

4.3 Camera

4.3.1. Camera Controller

Camera controller is written to handle camera movement reference to a Youtube Unity turtorial, **Character Creation (E08: third person camera)** made by Sebastian Lague, which teach you how to write a third person camera script. I added some features to the reference script to make it more functional, such as zooming feature and camera collision.

It changes camera position to follow player movement, so that character will not walk out of screen, player can always see their character in the screen.

Camera controller collect mouse input axis of horizontal and vertical, normalizing the vector, then rotate the camera toward input direction. Player can zoom in and out to change distance between camera and player character by scrolling the mouse wheel.

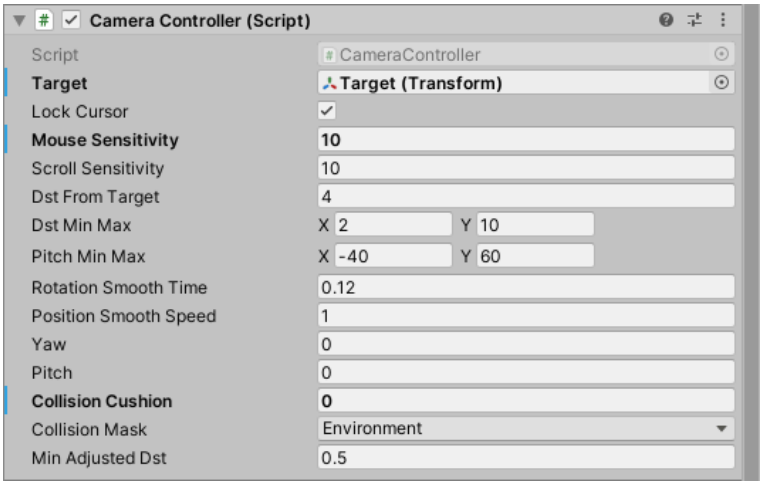

Figure 39 Camera Controller Components

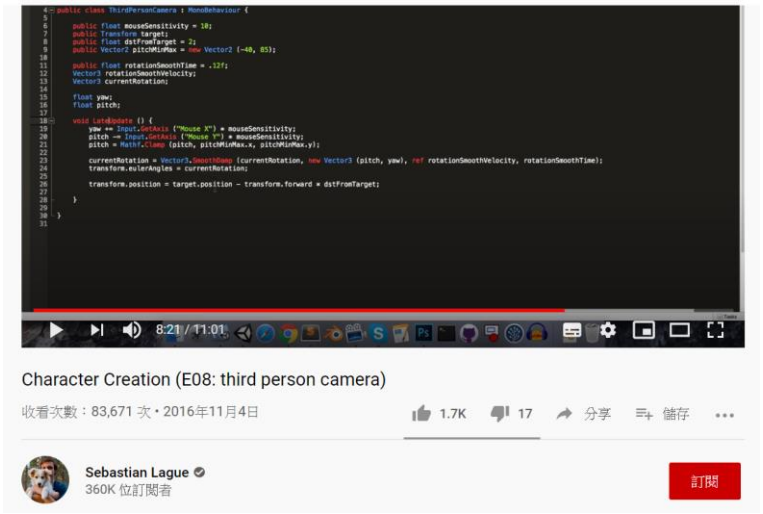

Figure 40 Camera - Youtube Turtorial

4.3.2. Camera Collision

Camera collision is implemented in order to prevent player seeing outside of the map. Rays are casted from player to camera, and camera distance will be adjusted to distance between wall and players, so that player can only see the interior of the map.

There is possibly for the clipping plane hit the wall, some parts of the wall will not be rendered if it hit the clipping plane then player is still able to see the outside of map. Therefore, five rays in total are cast toward different direction to avoid the clipping plane of camera hits the wall.

s

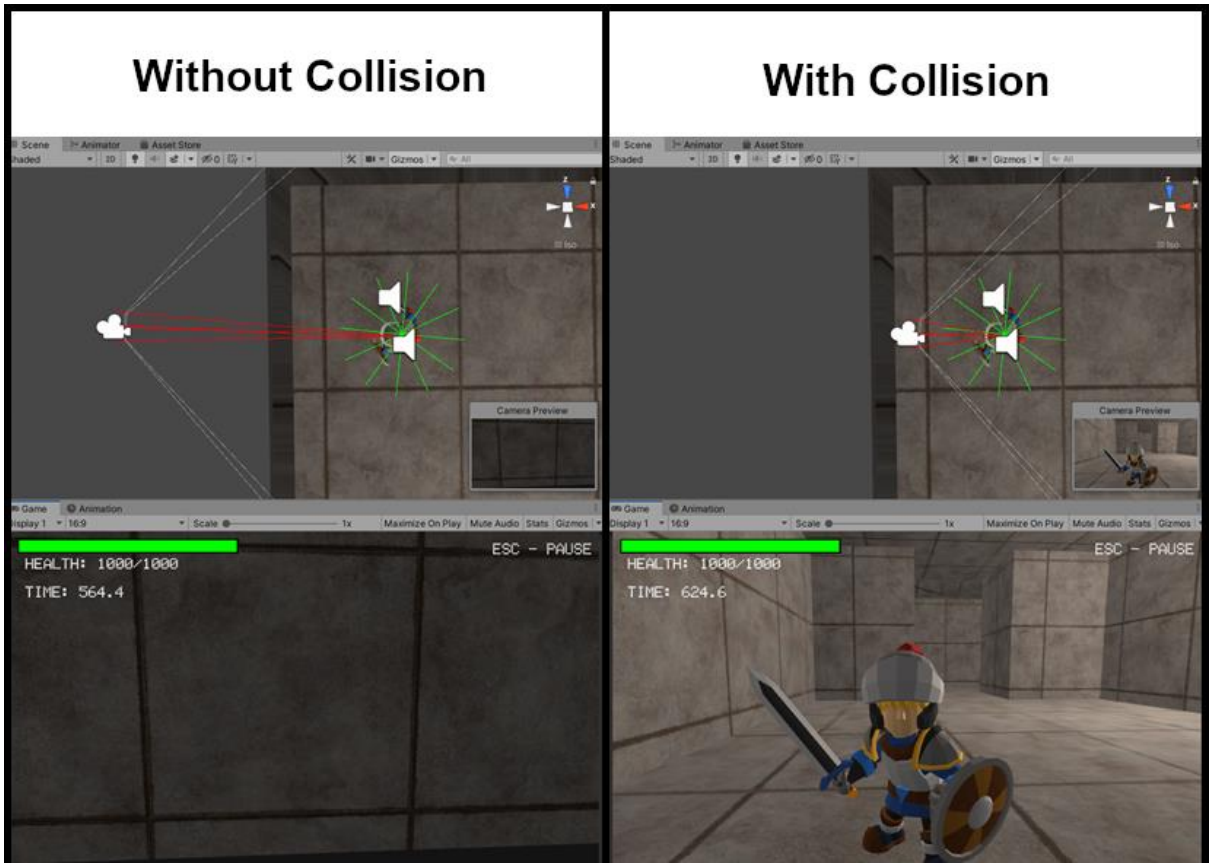

Figure 41 Camera Collision

Figure 42 Camera - Outside of Map

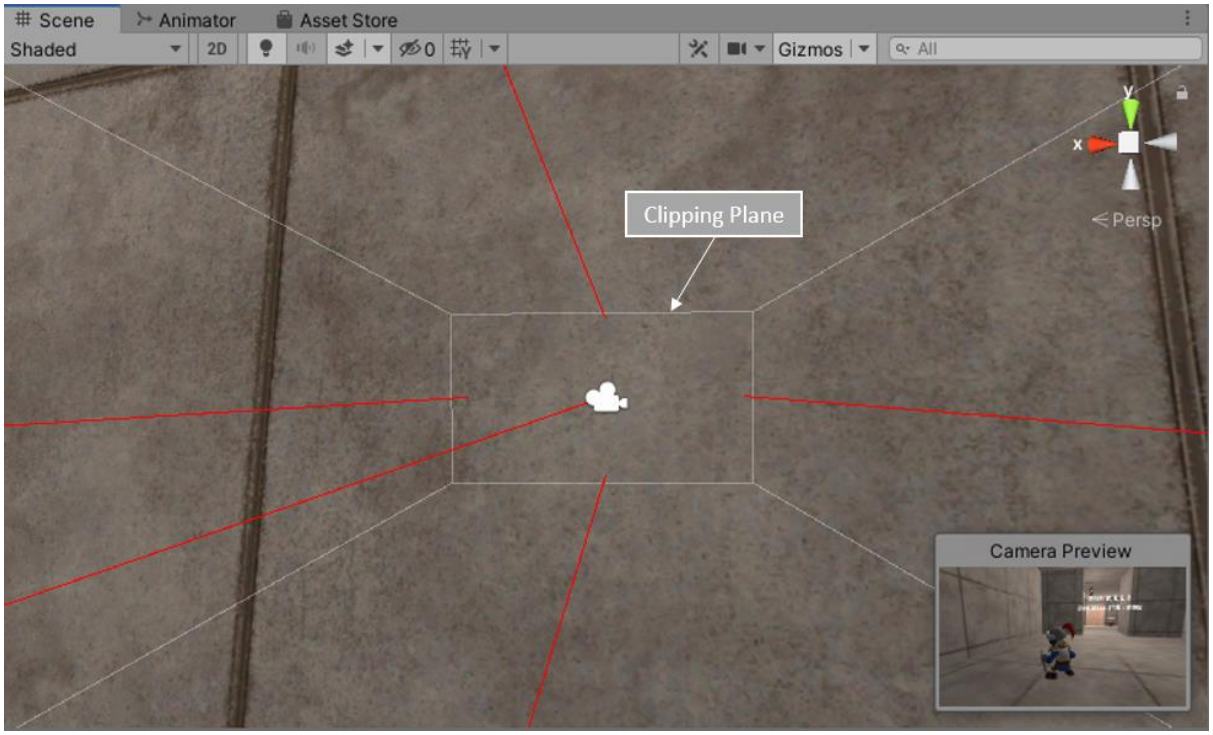

Figure 43 Camera - Clipping Plane

4.4 Enemy

4.4.1. Assets

Enemy model is also a very important part of the game, player spend half of the time to fight monsters in the gameplay. Enemy model must be good looking or player will lose interest on the game. 3D Models of the enemy were downloaded from unity assets store, they are also made by Dungeon Mason.

Figure 44 Enemy Assets – Monsters made by Dungeon Mason

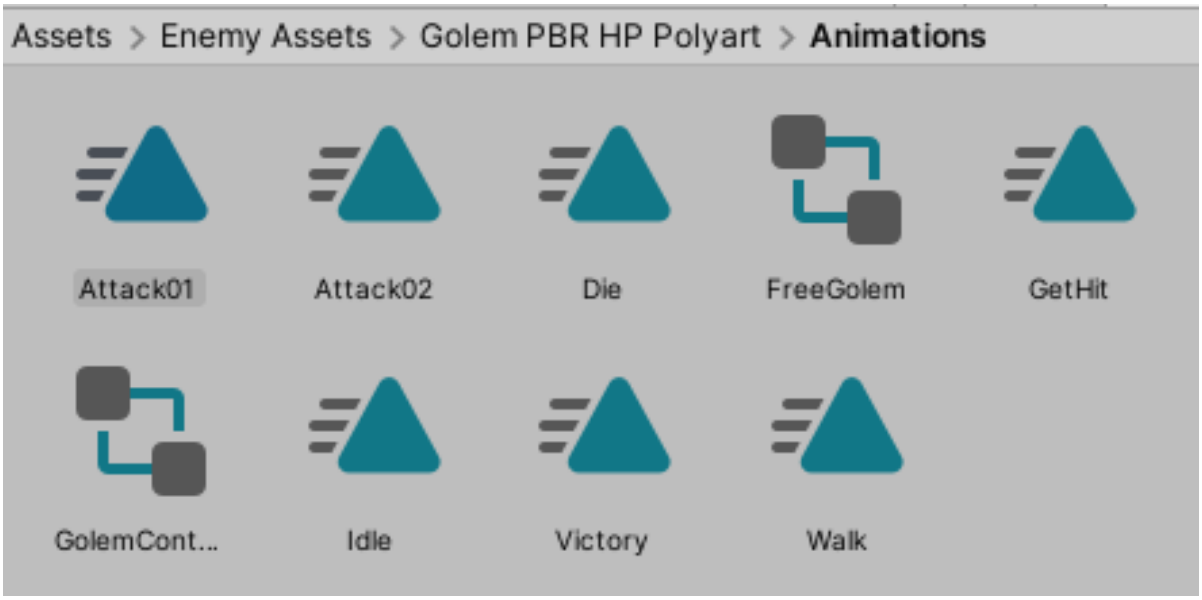

Figure 45 Enemy Assets – Animation Clip

I search a lot for free enemy assets in the assets store, but less of them have good graphic design. I need more of them in order to create various enemies. As a result, I decided to edit the texture of enemy model, changing its colour scale to create variant of enemy.

Figure 46 Enemy Assets - Variant

4.4.2 NavMeshAgent

According to Unity Manual, NavMeshAgent (Navigation Mesh Agent) components help you to create characters which avoid each other while moving towards their goal. Agents reason about the game world using the NavMesh.

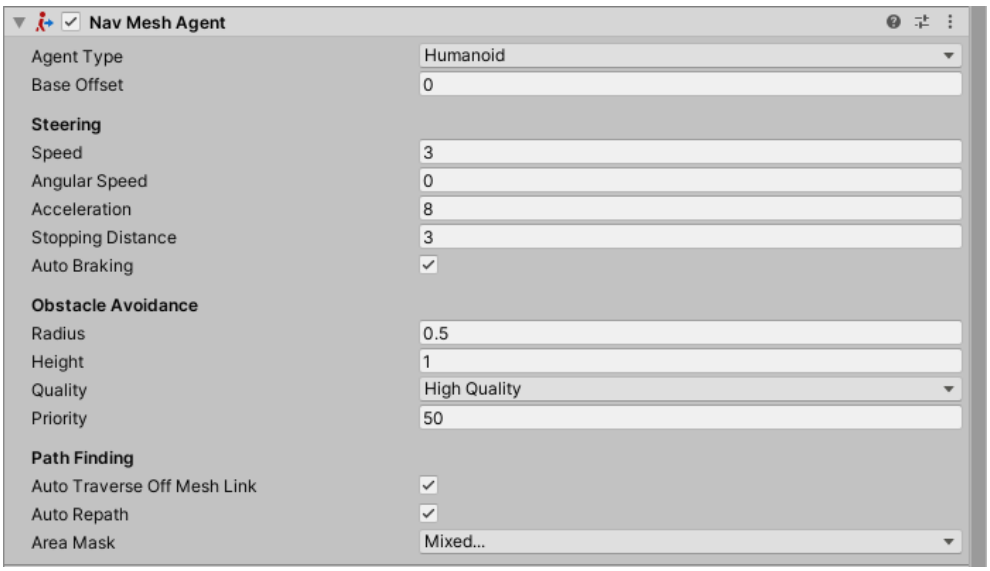

Figure 47 NavMeshAgent

To be brief, NavMeshAgent is a pathfinding AI, which can find path to target on the NavMesh, which determine the walkable area.

NavMeshAgent was used to enable enemy to move and track the player.

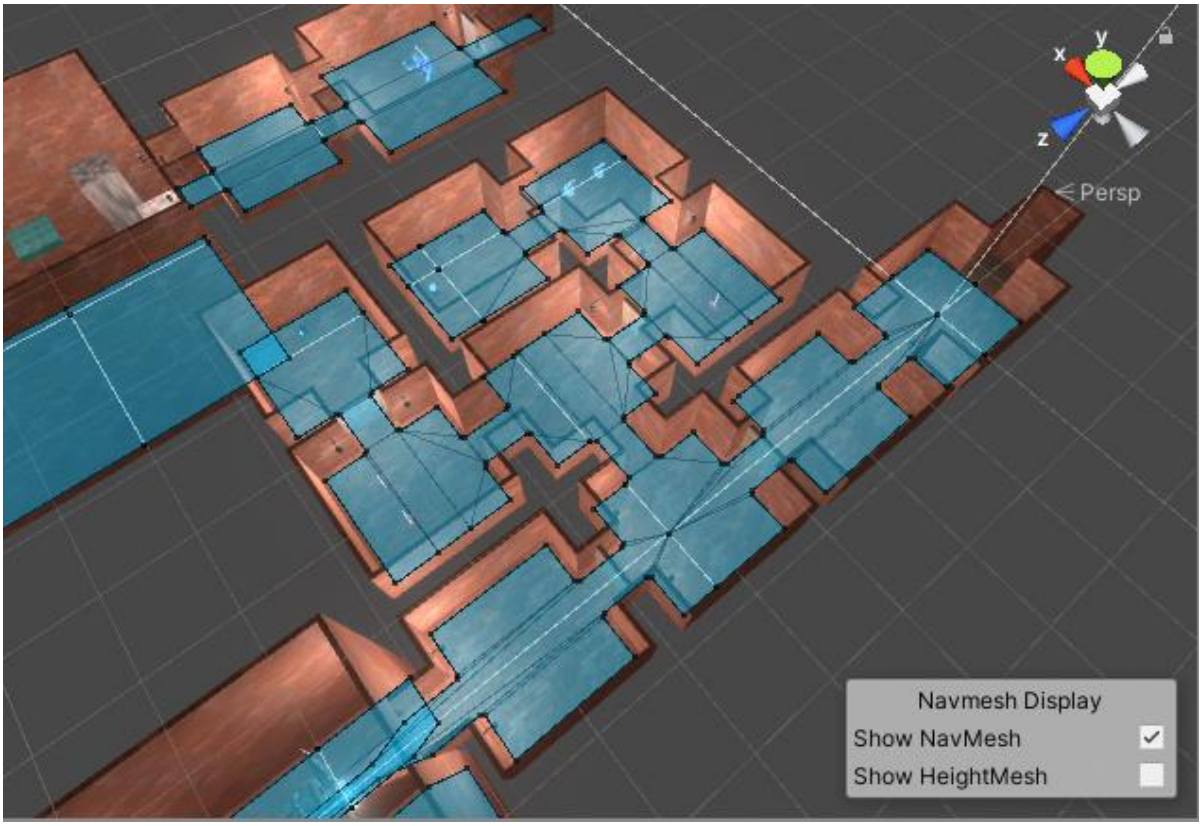

Figure 48 NavMesh

4.4.3. Enemy State

Enemy state is a similar to Player state but it has less factors. Enemy current health will be set to maximum when game starts and it will reduce because of player attack. The **Stun** parameter determine how often the enemy stun. If stun is set to 1, enemy will be stunned every time it gets hit. When enemy be stunned, a GetHit animation will be played and this will interrupt enemy attack animation. Strong enemies have less stun value so that they are less interruptible.

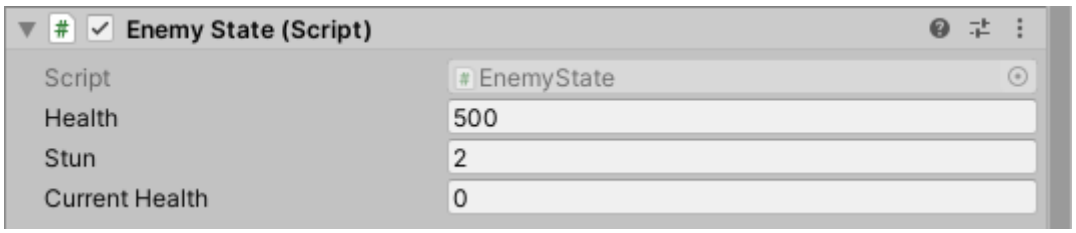

Figure 49 Enemy State

4.4.4. Enemy AI

In order to have a challenging game play, the enemy have to be smart. If the enemy just standing and keep attacking the air, player can easily kill the enemy by standing at its back. That's why the enemy AI is important, enemy must be able to trace the player. Therefore, an enemy AI was designed and implemented into my game, which contain four main state and will switch state on some situation.

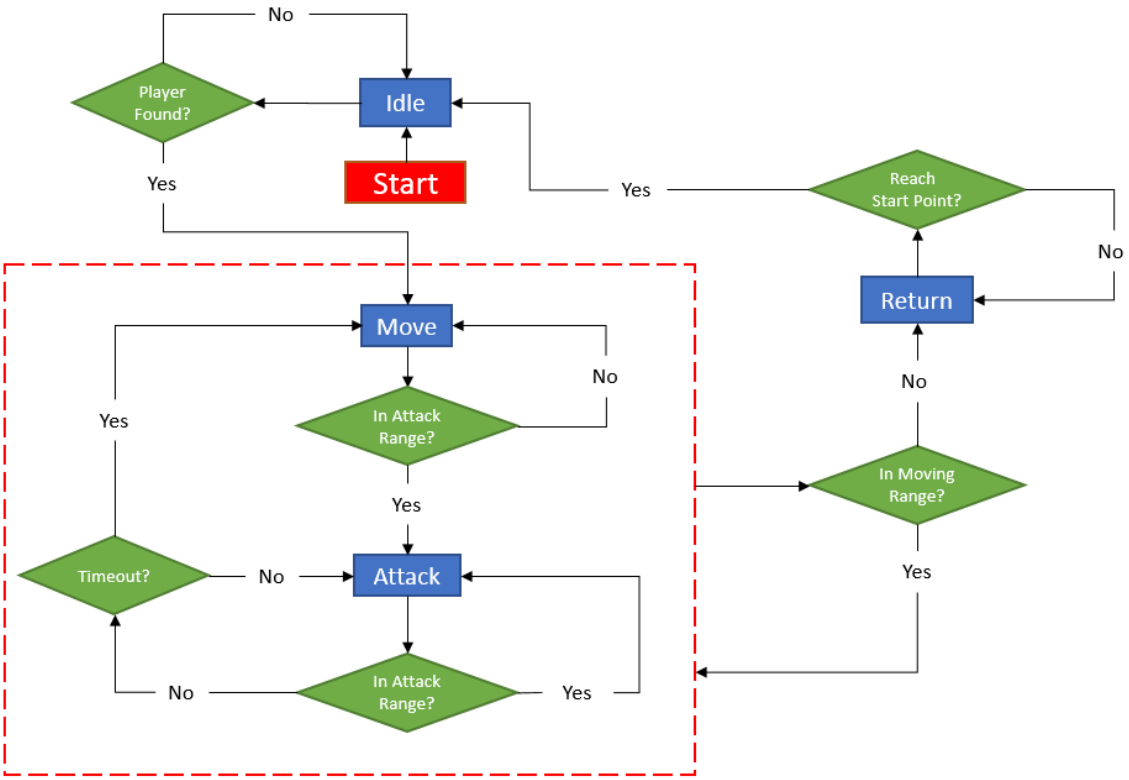

Figure 50 Enemy AI Diagram

Idle State:

Default state of all enemy, enemy will keep standing on their origin position until player is founded, "Player Found" is a boolean parameter that will be set to true when Player move close to the enemy. Note that Enemy is not able to recognize player if player stand behind the enemy. "Player Found" can also be triggered in other way through scripts, such as "When player enter the room".

Move State:

Once enemy find the player, the NavMeshAgent will be enabled and the enemy will start to approach the player. A rotation feature will also be enable in script so that enemy will keep facing the player. Actually NavMeshAgent also have the ability to face the target but I didn't use it because it will stop working when distance between player and enemy is smaller than the stopping distance. I want enemy keep rotate and facing the player even player is close to the enemy, so I implement this feature in the scripts but not using NavMeshAgent, it will move to the Attack state when player enter the attack range, which means the area that enemy is able to hit player during the attack animation, every type of monsters have different attack motion so different statement were made for each type of monsters.

Attack State:
Once player enter the attack range, enemy will stop chasing the player (stop moving and rotating) and start attacking, since player is now in the attack range, so player will get hit if character don't change its position. Player need to control the character to dodge the attack. When player leave the attack range, enemy will NOT switch state immediately, it will wait for 1-2 seconds. Player is able to attack the monster safely in this period of time. After a few seconds, enemy will switch to Move state and starting chasing player again. Note that the timer will be reset if player back to the attack range before it switches state. Also, after enemy get hit by player for a few times, a "GetHit" motion will be played and enemy will enter Move state immediately (Monsters get angry and start chasing player).

Return State:
Enemy has its maximum moving distance, enemy cannot move far away from its original position, otherwise player will die easily if they get chased by all enemies in map. Once player leave the maximum moving area, enemy will enter Return state and starting move back to its original position, where it was staying at when the game is started. It will back to Idle state when it reaches the original position.

4.4.6. Enemy Status

Enemy Name	Health	Stun	Short Attack		Long Attack	
			Damage	Knock down?	Damage	Knock down?
Slime	500	2	50	No	80	Yes
Turtle Shell	800	3	80	No	120	Yes
Grunt	1200	4	100	No	100	Yes
Big Grunt	2000	5	200	No	200	Yes
Gatekeeper	2000	7	200	Yes	100	No

Table 2 Enemy Status

4.4.7. HitBox
Same as player's hitbox, enemy's hitbox will only be enabled in certain frame during attack animation. Therefore, player will only get hit only when enemy is attacking.

4.5. UI

4.5.1. Health bar

A health bar UI is on the left upper corner of screen to show current health point of the player. The scripts keep checking the current health of player and change health bar length following health percentage.

4.5.2. Timer

Game will record the time when player press the start button, and keeps showing the timer on the left upper corner of screen.

4.5.3. Dialogue

Some dialogue was created to tell the story and give instruction for player Dialogues were stored in a string array and will be called when player hit the dialogues collider.

Figure 51 UI

5. Discussion

In the game developing process, the most challenging part is to create the game map in my opinion. At first, I tried not to use 3D modelling tool, and combine basic shapes object (Cube, Sphere) to create map environment. However, too many objects is need to create a single map, and the editor window is in a mess, also it is very restrictive, for example, a texture material have to be created for each objects in the map, this makes me have to create 30+ material for a single level. I can't make the map have the shape as I thought.
Then I understood I have to learn how to use 3D modelling tool to model the map. I spent so much time on watching tutorials on Youtube about Blender, learning how to create ground, walls and roof, assigning material to the model and unwarp the UV mesh. I spent a month on learning 3D modeling and finally a satisfying model was made.
You don't need to spend much time to make two once you know how to make one. I strongly suggest to create our own map using 3D modelling tool, it's worth learning.

In order to make the game looks natural and smooth, the program is tailor-made to fit the game. However, it is difficult to debug because they are very detailed. Once, I found a bug on the enemy AI and I spent 3 hours to debug it. My suggestion on debugging a game is to visualize the output, there are some method provided by Unity to print informational messages to the console, students are suggested to use them more to check if the output is correct. In 3D game, many of 3D vector will be used in the program. I suggested student to visualize them by Debug.DrawLine, which helps you to check if the vector is pointing the right direction.

6. Future improvement

After I let professor and my friends to play my game, they gave me a lot of useful feedback, some of them I already implemented in the game and some of them I don't have enough time to fix it. Therefore, I decide to make a to-do list for improving the game in the future.

1. **Add healing item:**
   One on a test player suggest me to add healing item so that he don't need to wait for the health to regenerate after the boss fight. To make the game challenging, the health regeneration rate is set to be low but I should give rewards to player for winning the boss fight. Healing Item or healing point will be added to the game in the next version.

2. **Add more type of enemy/ Reduce number of enemies:**
   Professor suggest me to add more type of enemies. Currently there are only 5 type of enemies in the game and on of it is the boss. Therefore, only 4 type of enemies will be place in the stage. Player may feel bored if they keep fighting the same monsters. Therefore, more monsters will be made in the future.

7. Conclusion

The game “Dungeon Adventure” was developed successfully. Many efforts have been made, from doing background research, learning skills, to finishing the game. I spent 500+ hours for this game. Although it is not perfect, I’m satisfied with the outcome.
I will keep improving my game and upload it to Github in the future.

Github: https://github.com/CodyLTH/Dungeon-Adventure

8. Appendix

Appendix 1: GameManager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    #region Singleton
    public static GameManager instance;
    void Awake()
    {
        if (instance == null)
        {
            DontDestroyOnLoad(gameObject);
            instance = this;
        }
        else if (instance != this)
        {
            Destroy(gameObject);
        }
    }
    #endregion

    public float maxHealth;
    public float currentHealth;
    public AudioSource normalBGM;
    public AudioSource clearBGM;
    public AudioSource bossBGM;
    public float startTime;

    public void SetTime()
    {
        startTime = Time.time;

    }
    public void playNormalBGM()
    {
        clearBGM.Stop();
        bossBGM.Stop();
        normalBGM.Play();
    }
    public void playClearBGM()
    {
        normalBGM.Stop();
        bossBGM.Stop();
        clearBGM.Play();

    }
    public void playBossBGM()
    {
        normalBGM.Stop();
        clearBGM.Stop();
        bossBGM.Play();

    }
}
```

Appendix 2: Player.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Player : MonoBehaviour
{
    #region Singleton
    public PlayerState state;
    public PlayerController controller;
    public static Player instance;
    void Awake()
    {

        if (instance == null)
        {
            instance = this;
            state = instance.GetComponent<PlayerState>();
            controller = instance.GetComponent<PlayerController>();
        }
        else
        {
            Destroy(gameObject);
        }
    }
    #endregion
}
```

Appendix 3: PlayerState.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PlayerState : MonoBehaviour
{
    Animator playerAnim;
    public Transform respawnPoint;
    public float healingRate = 10;
    public float maxHealth;
    public float currentHealth;
    public float hitInvincibleTime = 0.8f;
    public float knockInvincibleTime = 2.5f;
    public bool isInvincible = false;
    public bool isDead = false;

    public float dropDamage = 100f;

    public AudioSource getHitSE;

    void Start()
   {

        maxHealth = GameManager.instance.maxHealth;
        currentHealth = GameManager.instance.currentHealth;
        playerAnim = gameObject.GetComponentInChildren<Animator>();
        playerAnim.SetBool("IsGrounded", true);
    }

    private void Update()
    {
        if (isDead || isInvincible)
        {
            return;
        }
        currentHealth += healingRate * Time.deltaTime;
        currentHealth = Mathf.Clamp(currentHealth, 0, maxHealth);
    }

    public void TakeDamage(float amount, GameObject obj, bool knock)
    {
        if (isDead || isInvincible)
        {
            return;
        }
        isInvincible = true;
        currentHealth -= amount;

        if (obj != null)
        {
            var direction = obj.transform.position - transform.position;
            direction.y = 0;
            transform.rotation = Quaternion.LookRotation(direction, Vector3.up);
        }

        if (currentHealth <= 0)
        {
                currentHealth = 0;
                isDead = true;
                playerAnim.SetBool("IsDead", true);
        }
        if (knock)
```

```
        {
            Invoke("ResetInvincibleTimeFlag", knockInvincibleTime);
            Player.instance.controller.KnockedDownAnimation();
        }
        else
        {
            if (isDead)
            {
                Player.instance.controller.DieAnimation();
            }
            else
            {
                Player.instance.controller.GetHitAnimation();
                Invoke("ResetInvincibleTimeFlag", hitInvincibleTime);
            }
        }
        getHitSE.Play();
    }

    public void Heal(float ammount)
    {
        currentHealth += ammount;
        currentHealth = Mathf.Clamp(currentHealth, 0, maxHealth);
    }

    void ResetInvincibleTimeFlag()
    {
        if (!isDead)
        {
            isInvincible = false;
        }
    }

    void Die()
    {
        isDead = true;
        playerAnim.SetBool("IsDead", true);
        Debug.Log("Game Over");

    }

    void Respwan()
    {
        isInvincible = true;
        FadeUI.instance.StartFade("Respawn");
    }
```

```
    public void RespwanMove()
    {
        Invoke("ResetInvincibleTimeFlag", 1f);

        gameObject.SetActive(false);
        transform.position = respawnPoint.transform.position;
        transform.rotation = respawnPoint.transform.rotation;
        CameraController camController = Camera.main.gameObject.GetComponent<CameraController>();
        camController.pitch = 0;
        camController.yaw = transform.eulerAngles.y;
        Camera.main.transform.eulerAngles = transform.eulerAngles;
        gameObject.SetActive(true);
        playerAnim.SetBool("IsGrounded", true);
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Outside"))
        {
            if (!isInvincible)
            {
                currentHealth -= dropDamage;
            }
            if (currentHealth <= 0)
            {
                currentHealth = 0;
                Die();
            }
            else
            {
                Respwan();
            }
        }
    }
}
```

Appendix 4: PlayerController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerController : MonoBehaviour
{
    //Components
    Transform cameraT;
    Animator anim;
    CharacterController controller;
    //Settings
    public float walkSpeed = 3;
    public float runSpeed = 8;
    public float turnSmoothTime = 0.05f;
    public float gravity = -30f;
    public float jumpHeight = 5;
    public float speedSmoothTime = 0.1f;
    public float airControlPercent;
    //Variable
    float velocityY = 0;
    float turnSmoothVelocity;
    float speedSmoothVelocity;
    float currentSpeed;
    Vector2 inputDir;
    //States
    bool walking;
    public bool isAttacking = false;
    bool isMainState = false;
    bool isKnockedDown = false;
    bool isGettingHit = false;
    bool controllable = true;
    //Edge Detection
    Ray groundRay;
    Ray edgeRay;
    Vector3 edgeDetectMoveDir;
    public LayerMask collisionMask;

    void Start()
    {
        cameraT = Camera.main.transform;
        anim = GetComponentInChildren<Animator>();
        controller = GetComponent<CharacterController>();
    }
```

```
svoid Update()
{
    Vector3 velocity = transform.forward * currentSpeed + Vector3.up * velocityY;
    controller.Move(velocity * Time.deltaTime);
    GetCurrentState();

    if (controllable)
    {
        Vector2 input = new Vector2(Input.GetAxisRaw("Horizontal"), Input.GetAxisRaw("Vertical"));
        inputDir = input.normalized;
        walking = Input.GetKey(KeyCode.LeftShift);
    }

    EdgeDectect();
    Movement(inputDir, walking);

    if (!controller.isGrounded)
    {
        edgeDetectMoveDir = Vector3.zero;
        if (!Physics.Raycast(groundRay, 1.5f, collisionMask))
        {
            anim.SetBool("IsGrounded", false);
        }
        velocityY += Time.deltaTime * gravity;
        return;
    }

    velocityY = -3f;

    if (Physics.Raycast(groundRay, 0.8f, collisionMask))
    {
        anim.SetBool("IsGrounded", true);
    }

    if (controllable)
    {
        if (Input.GetMouseButtonDown(0))
        {
            Attack(inputDir);
        }
        if (Input.GetKeyDown(KeyCode.Space))
        {
            Jump();
        }
    }
    else
    {
        currentSpeed = 0;
        velocityY = -3f;
    }
}
```

```
    void Movement(Vector2 inputDir, bool walking)
    {
        if (Cursor.visible)
        {
            currentSpeed = Mathf.SmoothDamp(currentSpeed, 0, ref speedSmoothVelocity, speedSmoothTime);
            anim.SetFloat("Speed", 0);
            return;
        }

        if (Time.timeScale == 0 || Player.instance.state.isDead || !controllable)
        {
            return;
        }

        float targetSpeed = ((walking) ? walkSpeed : runSpeed) * inputDir.magnitude;
        currentSpeed = Mathf.SmoothDamp(currentSpeed, targetSpeed, ref speedSmoothVelocity,
speedSmoothTime);
        float targetRotation = Mathf.Atan2(inputDir.x, inputDir.y) * Mathf.Rad2Deg + cameraT.eulerAngles.y;

        if (inputDir != Vector2.zero)
        {
            transform.eulerAngles = Vector3.up * Mathf.SmoothDampAngle(transform.eulerAngles.y, targetRotation,
ref turnSmoothVelocity, GetModifiedSmoothTime(turnSmoothTime));
        }
        anim.SetFloat("Speed", targetSpeed*2/ runSpeed);
    }

    void Jump()
    {
        if (Time.timeScale == 0 || Player.instance.state.isDead || Cursor.visible || !controllable)
        {
            return;
        }
        float jumpVelocity = Mathf.Sqrt(-2 * gravity * jumpHeight);
        velocityY = jumpVelocity;
        anim.Play("JumpStart");
    }
```

```
void Attack(Vector2 inputDir)
{
    if (Time.timeScale == 0 || Player.instance.state.isDead || !controllable || Cursor.visible)
    {
        return;
    }
    anim.SetFloat("Speed", 0);

    if (Input.GetMouseButton(1))
    {
        transform.eulerAngles = Vector3.up * cameraT.eulerAngles.y;
        currentSpeed = 0;
        anim.Play("NormalAttack01");
    }
    else if (inputDir != Vector2.zero)
    {
        float targetRotation = Mathf.Atan2(inputDir.x, inputDir.y) * Mathf.Rad2Deg + cameraT.eulerAngles.y;
        transform.eulerAngles = Vector3.up * targetRotation;
        currentSpeed = 2;
        anim.Play("NormalAttack02");
    }
    else
    {
        currentSpeed = 0;
        anim.Play("NormalAttack01");
    }
}

public void GetHitAnimation()
{
    currentSpeed = 0;
    anim.Play("GetHit");
}

public void KnockedDownAnimation()
{
    anim.Play("KnockedDown");
    velocityY = 5f;
    currentSpeed = -10f;
}

public void DieAnimation()
{
    anim.Play("Die");
}

float GetModifiedSmoothTime(float smoothTime)
{
    if (controller.isGrounded)
    {
        return smoothTime;
    }

    if (airControlPercent == 0)
    {
        return float.MaxValue;
    }
    return smoothTime / airControlPercent;
}
```

```
    public void ResetCurrentSpeed()
    {
        if (inputDir == Vector2.zero)
        {
            currentSpeed = 0;
        }
    }

    void GetCurrentState()
    {
        isAttacking = anim.GetCurrentAnimatorStateInfo(0).IsName("NormalAttack01") ||
anim.GetCurrentAnimatorStateInfo(0).IsName("NormalAttack02");
        isKnockedDown = anim.GetCurrentAnimatorStateInfo(0).IsName("KnockedDown") ||
anim.GetCurrentAnimatorStateInfo(0).IsName("Recover");
        isGettingHit = anim.GetCurrentAnimatorStateInfo(0).IsName("GetHit");
        controllable = !(isAttacking || isKnockedDown || isGettingHit);
    }

    void EdgeDectect()
    {
        groundRay.origin = transform.position + new Vector3(0f, 0.5f, 0f);
        groundRay.direction = -transform.up;
        if (!Physics.Raycast(groundRay, 0.8f, collisionMask) && controller.isGrounded)
        {
            controller.SimpleMove(edgeDetectMoveDir * 20f);
        }
    }

    void OnControllerColliderHit(ControllerColliderHit hit)
    {
        if (!Physics.Raycast(groundRay, 0.8f, collisionMask) || controller.isGrounded)
        {
            edgeDetectMoveDir = Vector3.zero;
            float rayLength = 1.5f;
            edgeRay.origin = transform.position + Vector3.up * 0.2f;
            edgeRay.direction = transform.forward - Vector3.up * 0.1f;
            for (int i = 0; i < 12; i++)
            {
                edgeRay.direction = Quaternion.AngleAxis(30, Vector3.up) * edgeRay.direction;
                if (Physics.Raycast(edgeRay, rayLength))
                {
                    edgeDetectMoveDir -= new Vector3(edgeRay.direction.x, 0, edgeRay.direction.z);
                    Debug.DrawLine(edgeRay.origin, edgeRay.origin + edgeRay.direction * rayLength, Color.green);
                }
                else
                {
                    Debug.DrawLine(edgeRay.origin, edgeRay.origin + edgeRay.direction * rayLength, Color.red);
                }
            }
            if(edgeDetectMoveDir == Vector3.zero)
            {
                edgeDetectMoveDir = transform.forward;
            }
            edgeDetectMoveDir.Normalize();
        }
    }
}
```

Appendix 5: PlayerAnimationClipEvent.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerAnimationClipEvent : MonoBehaviour
{
    Weapon weapon;

    public AudioSource footstepSE;
    public AudioSource jumpSE;
    public AudioSource landSE;

    void Start()
    {
        weapon = GetComponentInChildren<Weapon>();
    }
    public void ResetPlayerSpeed()
    {
        Player.instance.controller.ResetCurrentSpeed();
    }

    public void PlayFootstepSE()
    {
        footstepSE.Play();
    }
    public void PlayJumpSE()
    {
        jumpSE.Play();
    }
    public void PlayLandSE()
    {
        landSE.Play();
    }

}
```

Appendix 6: Weapon.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Weapon : MonoBehaviour
{
    public float damage = 100f;
    public AudioSource hitSE;

    void OnTriggerEnter(Collider other)
    {
        if (Player.instance.controller.isAttacking && other.gameObject.CompareTag("EnemyCollider"))
        {
            EnemyState enemyState = other.gameObject.GetComponentInParent<EnemyState>();
            if(enemyState != null)
            {
                enemyState.TakeDamage(damage);
                hitSE.Play();
            }
        }
    }
}
```

Appendix 7: CameraController.cs

```
using UnityEngine;
using System.Collections.Generic;
using System.Collections;

public class CameraController : MonoBehaviour
{
    //Target
    public Transform target;

    //MouseSetting
    public bool lockCursor;
    public float mouseSensitivity = 10;
    public float scrollSensitivity = 10;

    //Camera Setting
    public float dstFromTarget = 5f;
    public Vector2 dstMinMax = new Vector2(4, 10);
    public Vector2 pitchMinMax = new Vector2(-30, 85);
    public float rotationSmoothTime = .12f;
    public float positionSmoothSpeed = 1f;

    //Variable
    Vector3 rotationSmoothVelocity;
    Vector3 currentRotation;
    public float yaw;
    public float pitch;

    //Collision
    public float collisionCushion = 0f;
    public LayerMask collisionMask;
    public Ray[] clipPoints = new Ray[5];
    float adjustedDst;
    public float minAdjustedDst = 0.5f;


    //Debug
    public bool collision = true;

    void Start()
    {
        target = Player.instance.gameObject.transform.Find("Target");
        adjustedDst = dstFromTarget;
        Cursor.lockState = CursorLockMode.Locked;
        Cursor.visible = false;
    }
```

```
void LateUpdate()
    {
        if(Time.timeScale == 0 || Cursor.visible)
        {
            yaw = currentRotation.y;
            pitch = currentRotation.x;
            CameraCollision();
            return;
        }
        dstFromTarget -= Input.GetAxis("Mouse ScrollWheel") * scrollSensitivity;
        dstFromTarget = Mathf.Clamp(dstFromTarget, dstMinMax.x, dstMinMax.y);
        yaw += Input.GetAxis("Mouse X") * mouseSensitivity;
        pitch -= Input.GetAxis("Mouse Y") * mouseSensitivity;
        pitch = Mathf.Clamp(pitch, pitchMinMax.x, pitchMinMax.y); ;

        currentRotation = Vector3.SmoothDamp(currentRotation, new Vector3(pitch, yaw), ref
rotationSmoothVelocity, rotationSmoothTime);
        transform.eulerAngles = currentRotation;

        CameraCollision();

    }

    void CameraCollision()
    {
        RaycastHit RayHit;
        Vector3 desiredPosition;
        clipPoints[0] = new Ray(target.transform.position, transform.position - target.transform.position);
        clipPoints[1] = new Ray(target.transform.position, transform.position + transform.up * 0.2f -
target.transform.position);
        clipPoints[2] = new Ray(target.transform.position, transform.position - transform.up * 0.2f -
target.transform.position);
        clipPoints[3] = new Ray(target.transform.position, transform.position + transform.right * 0.35f -
target.transform.position);
        clipPoints[4] = new Ray(target.transform.position, transform.position - transform.right * 0.35f -
target.transform.position);

        adjustedDst = dstFromTarget;
        foreach (Ray ray in clipPoints)
        {
            if (Physics.Raycast(ray, out RayHit, dstFromTarget, collisionMask))
            {
                float distance = RayHit.distance - collisionCushion;
                if (distance < adjustedDst && collision)
                {
                    adjustedDst = distance;
                }
            }
            Debug.DrawLine(ray.origin, ray.origin + ray.direction * adjustedDst, Color.red);
        }
        adjustedDst = Mathf.Max(adjustedDst, minAdjustedDst);
        desiredPosition = target.position - transform.forward * adjustedDst;
        transform.position = Vector3.Lerp(transform.position, desiredPosition, positionSmoothSpeed);

    }
}
```

Appendix 8: RespawnPoint.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RespawnPoint : MonoBehaviour
{
    bool flag = true;

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject && flag)
        {
            Debug.Log("Update Respawn Point");
            Player.instance.state.respawnPoint = gameObject.transform;
            flag = false;
        }
    }
}
```

Appendix 9: EnemyState.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class EnemyState : MonoBehaviour
{
    public float health = 500f;
    public float stun = 1;
    public float currentHealth;

    float stunCount = 0;
    EnemyController enemyContoller;

    void Start()
  {
        currentHealth = health;
        stunCount = 0;
        enemyContoller = GetComponent<EnemyController>();
    }

    public void TakeDamage(float amount)
    {
        if(enemyContoller.playerFound == false)
        {
            enemyContoller.playerFound = true;
            enemyContoller.TakeDamageAnimation();
        }
        currentHealth -= amount;
        Debug.Log("Current HP = " + currentHealth);
        if (currentHealth <= 0)
        {
            currentHealth = 0;
            enemyContoller.DieAnimation();
            Invoke("Destroy", 2f);
        }
        else
        {
            stunCount++;
            if ( stunCount >= stun)
            {
                enemyContoller.TakeDamageAnimation();
                stunCount = 0;
            }
        }
    }

    void Destroy()
    {
        Destroy(gameObject);
    }

}
}
```

Appendix 10: EnemyController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class EnemyController : MonoBehaviour
{
    // Start is called before the first frame update


    [SerializeField] protected float damage = 100f;
    [SerializeField] protected float lookBackTime = 3f;
    [SerializeField] protected float turnSmoothTime = 3;
    [SerializeField] protected float maxMovingDistance = 20;
    [SerializeField] protected float lookDistance = 10;
    [SerializeField] protected float stopDistance = 3;
    [SerializeField] protected float shortRange = 2;
    [SerializeField] protected float longRange = 4;


    float agentSpeed;
    float agentAccel;


    protected Transform target;
    protected NavMeshAgent agent;
    protected Animator anim;

    public bool playerFound = false;

    string mode = "Idle";

    bool isDead = false;
    bool isAttacking = false;
    bool isAnimation = false;


    Quaternion rotation;
    float time;
    protected float distance;
    protected float angle;
    protected bool willKnockDown = false;
    bool OutOfRange;

    Vector3 startPoint;
    Quaternion startRotation;



    protected void Start()
    {
        target = Player.instance.transform;
        agent = GetComponent<NavMeshAgent>();
        anim = GetComponentInChildren<Animator>();
        time = Time.time;
        playerFound = false;
        mode = "Idle";
        startPoint = transform.position;
        startRotation = transform.rotation;
        agent.stoppingDistance = stopDistance;
    }
```

```
    void Update()
    {
        if (isDead)
        {
            time = Time.time;
            return;
        }

        distance = Vector3.Distance(target.position, transform.position);
        Vector3 targetDir = target.position - transform.position;
        targetDir.y = 0;
        targetDir.Normalize();
        angle = Vector3.SignedAngle(targetDir, transform.forward, Vector3.up);
        rotation = Quaternion.LookRotation(targetDir, Vector3.up);
        isAttacking = anim.GetCurrentAnimatorStateInfo(0).IsName("ShortAttack") ||
anim.GetCurrentAnimatorStateInfo(0).IsName("LongAttack");
        isAnimation = anim.GetCurrentAnimatorStateInfo(0).IsName("GetHit") ||
anim.GetCurrentAnimatorStateInfo(0).IsName("Die") || anim.GetCurrentAnimatorStateInfo(0).IsName("Found");

        if (mode != "Idle" && (Vector3.Distance(target.position, startPoint) > maxMovingDistance ||
Player.instance.state.isDead))
        {
            mode = "Return";
            ReturnMode();
            return;
        }
        if (isAttacking || isAnimation)
        {
            agent.isStopped = true;
            return;
        }
        if (mode == "Move")
        {
            MoveMode();
        }
        else if (mode == "Attack")
        {
            AttackMode();
        }
        else if (mode == "Idle")
        {
            IdleMode();
        }
        else if (mode == "Return")
        {
            ReturnMode();
        }
    }
```

```
void IdleMode()
{
    agent.isStopped = true;
    anim.SetFloat("Speed", 0);
    agent.SetDestination(target.position);
    if (distance < lookDistance && Mathf.Abs(angle) < 90)
    {
        playerFound = true;
    }
    if (playerFound)
    {
        anim.Play("Found");
        mode = "Move";
        time = Time.time;
    }
}

void MoveMode()
{
    agent.isStopped = false;
    anim.SetFloat("Speed", 1);
    transform.rotation = Quaternion.Slerp(transform.rotation, rotation, Time.deltaTime * turnSmoothTime);
    agent.SetDestination(target.position);
    if (!isAttacking && !isAnimation && inAttackArea() && distance <= stopDistance)
    {
        mode = "Attack";
        time = Time.time;
    }
}

void AttackMode()
{
    agent.isStopped = true;
    anim.SetFloat("Speed", 0);
    if (!isAttacking && inAttackArea() && distance <= longRange)
    {
        time = Time.time;
        Attack();
    }
    if (!isAttacking && !isAnimation && (Time.time - time > lookBackTime || distance > longRange))
    {
        mode = "Move";
        time = Time.time;
    }
}


public virtual bool inAttackArea()
{
    if (angle > 30)
    {
        return false;
    }
    return true;
}
```

```
    public virtual void Attack()
    {
        if (distance < shortRange)
        {
            anim.SetTrigger("ShortAttack");
            anim.ResetTrigger("LongAttack");
            willKnockDown = false;
        }
        else if (distance < longRange)
        {
            anim.SetTrigger("LongAttack");
            anim.ResetTrigger("ShortAttack");
            willKnockDown = true;
        }
    }

    void ReturnMode()
    {
        agent.isStopped = false;
        if (Player.instance.state.isDead)
        {
            restart();
            return;
        }
        Vector3 targetDir = startPoint - transform.position;
        targetDir.y = 0;
        targetDir.Normalize();
        if (targetDir != Vector3.zero)
        {
            rotation = Quaternion.LookRotation(targetDir, Vector3.up);
            transform.rotation = Quaternion.Slerp(transform.rotation, rotation, Time.deltaTime * turnSmoothTime *
4);
        }
        if (agent.stoppingDistance > 0)
        {
            agent.SetDestination(startPoint);
            anim.SetFloat("Speed", 1);
            agentSpeed = agent.speed;
            agentAccel = agent.acceleration;
            agent.speed = 10f;
            agent.acceleration = 20f;
            agent.stoppingDistance = 0f;
        }
        else if (Vector3.Distance(transform.position, startPoint) < 0.2f)
        {
            agent.velocity = Vector3.zero;
            agent.speed = agentSpeed;
            agent.stoppingDistance = stopDistance;
            agent.SetDestination(target.position);
            restart();
        }
    }
```

```
    void restart()
    {
        anim.SetFloat("Speed", 0);
        playerFound = false;
        time = Time.time;
        transform.position = startPoint;
        transform.rotation = startRotation;
        GetComponent<EnemyState>().currentHealth = GetComponent<EnemyState>().health;
        mode = "Idle";
    }

    public void TakeDamageAnimation()
    {
        agent.isStopped = true;
        anim.Play("GetHit");
        mode = "Move";
    }

    public void DieAnimation()
    {
        agent.isStopped = true;
        anim.Play("Die");
        isDead = true;
    }

    public virtual void OnDrawGizmosSelected()
    {
        Gizmos.color = Color.white;
        if (playerFound)
        {
            Gizmos.DrawWireSphere(startPoint, maxMovingDistance);
        }
        else
        {
            Gizmos.DrawWireSphere(transform.position, maxMovingDistance);
        }
        Gizmos.color = Color.cyan;
        Gizmos.DrawWireSphere(transform.position, lookDistance);
        Gizmos.color = Color.green;
        Gizmos.DrawWireSphere(transform.position, stopDistance);
        Gizmos.color = Color.red;
        Gizmos.DrawWireSphere(transform.position, shortRange);
        Gizmos.color = Color.yellow;
        Gizmos.DrawWireSphere(transform.position, longRange);
    }

    private void OnTriggerEnter(Collider other)
    {
        isAttacking = anim.GetCurrentAnimatorStateInfo(0).IsName("ShortAttack") ||
anim.GetCurrentAnimatorStateInfo(0).IsName("LongAttack");
        if (other.gameObject == Player.instance.gameObject && isAttacking)
        {
            Player.instance.state.TakeDamage(damage, gameObject, willKnockDown);
            Debug.Log(damage);
        }
    }

}
```

Appendix 11: SlimeController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SlimeController : EnemyController
{
    public float shortDamage;
    public float longDamage;

    public override void Attack()
    {
        if (distance < shortRange)
        {
            damage = shortDamage;
            anim.SetTrigger("ShortAttack");
            willKnockDown = false;
        }
        else if (distance < longRange)
        {
            damage = longDamage;
            anim.SetTrigger("LongAttack");
            willKnockDown = true;
        }
    }

    public override bool inAttackArea()
    {
        if (Mathf.Abs(angle) > 20)
        {
            return false;
        }
        return true;
    }
}
```

Appendix 12: GruntController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GruntController : EnemyController
{
    public override void Attack()
    {
        if (Mathf.Abs(angle) < 10)
        {
            anim.SetTrigger("ShortAttack");
            willKnockDown = false;
        }
        else
        {
            anim.SetTrigger("LongAttack");
            willKnockDown = true;
        }
    }

    public override bool inAttackArea()
    {
        if (Mathf.Abs(angle) > 25)
        {
            return false;
        }
        return true;
    }
}
```

Appendix 13: GateKeeperController.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class GateKeeperController : EnemyController
{

    public GameObject rockPrefab;

    public override void Attack()
    {
        if (distance < shortRange)
        {
            anim.SetTrigger("ShortAttack");
            willKnockDown = true;
        }
        else if (distance < longRange)
        {
            anim.SetTrigger("LongAttack");
            willKnockDown = true;
        }
    }

    public override bool inAttackArea()
    {
        if (angle > 40 || angle < -10)
        {
            return false;
        }
        return true;
    }

    public void ThrowRock()
    {
        GameObject obj = Instantiate(rockPrefab, rockPrefab.gameObject.transform.position,
rockPrefab.gameObject.transform.rotation);
        obj.GetComponent<GateKeeperRock>().forward = transform.forward;
        obj.GetComponent<GateKeeperRock>().forward = transform.forward;
        obj.SetActive(true);

    }
}
```

Appendix 14: GateKeeperRock.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GateKeeperRock : MonoBehaviour
{
    Rigidbody rb;

    public float damage;
    public float speed = 0.3f;
    public Vector3 forward;
    public Vector3 direction;
    float count = 0;
    bool flag = true;

    void Start()
    {
        rb = GetComponent<Rigidbody>();
        direction = Player.instance.transform.position - transform.position + Vector3.up;
        direction.Normalize();

        Vector3 normal = Vector3.Cross(forward, direction);
        float angle = Vector3.Angle(forward, direction);
        angle = Mathf.Clamp(angle, 0, 60);
        Vector3 dir = Quaternion.AngleAxis(angle, normal) * forward;
        rb.velocity = dir * speed;
    }
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject && flag)
        {
            Player.instance.state.TakeDamage(damage, gameObject, false);
            flag = false;
            Invoke("DestroyObject", 0.8f);
        }
    }

    private void OnCollisionEnter(Collision collision)
    {
        if(collision.gameObject != Player.instance.gameObject)
        {
            flag = false;
        }
        Invoke("DestroyObject", 1.5f);
    }

    private void DestroyObject()
    {
        Destroy(gameObject);
    }
}
```

Appendix 15: EnemyHealthBar.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class EnemyHealthBar : MonoBehaviour
{
    // Start is called before the first frame update

    public Image healthBar;

    float max;
    float current;

    void Start()
    {
        max = GetComponentInParent<EnemyState>().health;
    }

    // Update is called once per frame
    void LateUpdate()
    {
        transform.rotation = Camera.main.transform.rotation;
        current = GetComponentInParent<EnemyState>().currentHealth;
        healthBar.fillAmount = current / max;
        healthBar.color = Color.HSVToRGB(current / (3 * max), 1, 1);
        if(healthBar.fillAmount == 0)
        {
            gameObject.SetActive(false);
        }
    }
}
```

Appendix 16: GateKeyTrigger.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class GateKeyTrigger : MonoBehaviour
{
    Animator anim;
    AudioSource sound;
    bool opened;
    MeshRenderer meshRenderer;

    // Start is called before the first frame update
    void Start()
    {
        opened = false;
        anim = GetComponentInParent<Animator>();
        sound = GetComponentInParent<AudioSource>();
        meshRenderer = GetComponentInParent<MeshRenderer>();
        meshRenderer.material.color = Color.red;
        meshRenderer.material.SetColor("_EmissionColor", Color.red * 1f);
    }

    void GateOpen()
    {
        if (!opened)
        {
            opened = true;
            anim.SetBool("Open", true);
            sound.Play();
            meshRenderer.material.color = Color.green;
            meshRenderer.material.SetColor("_EmissionColor", Color.green * 1f);
        }

    }
    public void GateClose()
    {
        if (opened)
        {
            opened = false;
            anim.SetBool("Open", false);
            meshRenderer.material.color = Color.red;
            meshRenderer.material.SetColor("_EmissionColor", Color.red * 1f);
        }
    }

    private void OnTriggerStay(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject)
        {
            if (Input.GetKey(KeyCode.E))
            {
                GateOpen();
            }
        }
    }
}
```

Appendix 17: GateEnemyTrigger.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI;
public class GateEnemyTrigger : MonoBehaviour
{
    public static int enemyCount = 1;
    public GameObject[] enemies = new GameObject[enemyCount];

    Animator anim;
    AudioSource sound;
    NavMeshObstacle obstacle;
    bool opened;


    // Start is called before the first frame update
    void Start()
  {
        opened = false;
        obstacle = GetComponentInParent<NavMeshObstacle>();
        anim = GetComponentInParent<Animator>();
        sound = GetComponentInParent<AudioSource>();
    }

  // Update is called once per frame
  void Update()
  {
        if (opened)
        {
            return;
        }
        foreach (GameObject obj  in enemies)
        {
            if(obj != null)
            {
                return;
            }
        }
        opened = true;
        GateOpen();
    }

    void GateOpen()
    {
        anim.SetBool("Open", true);
        if(obstacle != null) obstacle.enabled = false;
        sound.Play();
    }

}
```

Appendix 18: GateButtonTrigger.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GateButtonTrigger : MonoBehaviour
{
    Animator anim;
    AudioSource sound;
    bool opened;
    MeshRenderer meshRenderer;

    // Start is called before the first frame update
    void Start()
    {
        opened = false;
        anim = GetComponentInParent<Animator>();
        sound = GetComponentInParent<AudioSource>();
        meshRenderer = GetComponentInParent<MeshRenderer>();
        meshRenderer.material.color = Color.red;
        meshRenderer.material.SetColor("_EmissionColor", Color.red);
    }

    void GateOpen()
    {
        if (!opened)
        {
            opened = true;
            anim.SetBool("Open", true);
            sound.Play();
            meshRenderer.material.color = Color.green;
            meshRenderer.material.SetColor("_EmissionColor", Color.green);
        }
    }

    private void OnTriggerEnter(Collider other)
    {
        GateOpen();
    }
}
```

Appendix 19: BossRoomTrigger.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BossRoomTrigger : MonoBehaviour
{
    public GateKeyTrigger script;
    public static int enemyNumber;
    public EnemyController[] enemies = new EnemyController[enemyNumber];

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject)
        {
            if (script != null)
            {
                script.GateClose();
            }
            foreach(EnemyController ec in enemies)
            {
                ec.playerFound = true;
            }
        }
    }
}
```

Appendix 20: RollingStone.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class RollingStone : MonoBehaviour
{
    Rigidbody rb;

    public float damage;
    public float speed = 0.3f;
    Vector3 direction;


    void Start()
    {
        direction = transform.forward;
        rb = GetComponent<Rigidbody>();
    }

    void FixedUpdate()
    {
        rb.AddForce(direction * speed);
    }


    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject)
        {
            Player.instance.state.TakeDamage(damage, gameObject, true);

        }
        else if (other.gameObject.CompareTag("Outside"))
        {
            Destroy(gameObject);
        }
    }

}
```

Appendix 21: GenerateStone.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class GenerateStone : MonoBehaviour
{
    public GameObject stonePrefab;
    public float period = 2f;

    // Start is called before the first frame update
    void Start()
  {
        InvokeRepeating("Generate", period, period);
    }

  // Update is called once per frame
  void Update()
  {

  }

    void Generate()
    {
        GameObject prefab = Instantiate(stonePrefab, transform.position, transform.rotation);
        prefab.transform.SetParent(gameObject.transform);
    }

}
```

Appendix 22: Trap_Axe.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Trap_Axe : MonoBehaviour
{
    public float damage;
    public GameObject hitCenter;

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject)
        {
            Player.instance.state.TakeDamage(damage, hitCenter, true);
        }
    }
}
```

Appendix 23: Trap_Cutter.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Trap_Cutter : MonoBehaviour
{
    public float damage;
    Collider[] colChildren;

    // Start is called before the first frame update
    void Start()
  {
        colChildren = GetComponentsInChildren<Collider >();
    }

  // Update is called once per frame
  void Update()
  {
        foreach (Collider collider in colChildren)
        {
            if (Player.instance.state.isInvincible)
            {
                collider.enabled = false;
            }
            else
            {
                collider.enabled = true;
            }
        }


    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject)
        {
            Player.instance.state.TakeDamage(damage, gameObject, true);
        }
    }
}
```

Appendix 24: DroppingPlane.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DroppingPlane : MonoBehaviour
{
    Animator anim;
    void Start()
    {
        anim = GetComponentInParent<Animator>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject)
        {
            anim.SetTrigger("Drop");
        }
    }
}
```

Appendix 25: HealingItem.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class HealingItem : MonoBehaviour
{
    public float ammount = 100f;

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject)
        {
            Player.instance.state.Heal(ammount);
            Destroy(gameObject);
        }
    }
}
```

Appendix 26: HealthBar.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class HealthBar : MonoBehaviour
{
    // Start is called before the first frame update

    public Image healthBar;
    public TextMeshProUGUI textMesh;

    float max;
    float current;

    void Start()
    {
        max = GameManager.instance.maxHealth;
    }

    // Update is called once per frame
    void LateUpdate()
   {
        current = Player.instance.state.currentHealth;
        healthBar.fillAmount = current / max;
        textMesh.text = "Health: " + (int)current + "/" + (int)max;
        healthBar.color = Color.HSVToRGB(current/(3*max), 1, 1);

    }
}
```

Appendix 27: Timer.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class Timer : MonoBehaviour
{
    public TextMeshProUGUI textMesh;

    float time;
    double time2;

    // Update is called once per frame
    void LateUpdate()
    {
        time = Time.time - GameManager.instance.startTime;
        if(time < 10000)
        {
            textMesh.text = "Time: " + time.ToString("F1"); ;
        }
        else
        {
            textMesh.text = "Time: N/A";
        }
    }
}
```

Appendix 28: StartMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class StartMenu : MonoBehaviour
{
    // Start is called before the first frame update

    public void PlayGame()
    {
        FadeUI.instance.StartFade("NextLevel");
        GameManager.instance.SetTime();
    }

    public void Quit()
    {
        Debug.Log("Quit");
        Application.Quit();
    }

}
```

Appendix 29: PauseMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
using TMPro;

public class PauseMenu : MonoBehaviour
{

    public static bool GameIsPauesed = false;

    public GameObject pauseMenuUI;
    public GameObject fadeImage;
    public GameObject gameOverUI;
    public TextMeshProUGUI textMesh;

    void Update()
  {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GameIsPauesed)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
  }

    public void Resume()
    {
        Time.timeScale = 1;
        pauseMenuUI.SetActive(false);
        GetComponent<CursorLock>().LockCursor();
        textMesh.text = "Esc - Pause";
        GameIsPauesed = false;
    }

    public void Pause()
    {
        if (fadeImage.activeSelf ||gameOverUI.activeSelf )
        {
            return;
        }
        Time.timeScale = 0;
        pauseMenuUI.SetActive(true);
        GetComponent<CursorLock>().ShowCursor();
        textMesh.text = "Esc - Resume";
        GameIsPauesed = true;
    }

    public void Menu()
    {
        Time.timeScale = 1;
        SceneManager.LoadScene("StartMenu");
    }
```

```
public void Quit()
    {
        Time.timeScale = 1;
        Debug.Log("Quit Game");
        Application.Quit();
    }

}
```

Appendix 30: FadeUI.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
public class FadeUI : MonoBehaviour
{
    #region Singletons

    public Image fadeImage;
    public GameObject canvas;
    public static FadeUI instance;
    void Awake()
    {
        if (instance == null)
        {
            instance = this;
        }
        else if (instance != this)
        {
            Destroy(gameObject);
        }
    }
    #endregion

    string mode;

    public void StartFade(string m)
    {
        mode = m;
        Animator anim = GetComponent<Animator>();
        if (m == "Goal")
        {
            fadeImage.color = Color.white;
        }
        else
        {
            fadeImage.color = Color.black;
        }
        anim.Play("FadeIn");
    }

    public void FadeOut()
    {
        Animator anim = GetComponent<Animator>();
        anim.Play("FadeOut", -1, 0f);
    }



    void FadeEvent()
    {
        switch (mode)
        {
            case "Respawn":
                Player.instance.state.RespwanMove();
                break;
            case "Die":
                canvas.GetComponent<GameOver>().showUI();
                break;
            case "NextLevel":
```

```
                SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
                break;
            case "Goal":
                SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
                break;
            case "Menu":
                SceneManager.LoadScene("StartMenu");
                break;
        }
    }
}
```

Appendix 31: Dialogue.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class Dialogue : MonoBehaviour
{

    public GameObject dialogueUI;

    TextMeshProUGUI textMesh;
    string[] dialogue;
    int index;

    private void Start()
    {
        textMesh = dialogueUI.GetComponentInChildren<TextMeshProUGUI>();
    }

    public void DisplayDialogue(string[] d)
    {
        dialogue = d;
        if(dialogue.Length <= 0)
        {
            return;
        }
        dialogueUI.SetActive(true);
        index = 0;
        textMesh.text = dialogue[0];
        GetComponent<CursorLock>().ShowCursor();
    }

    public void NextDialogue()
    {
        index++;
        if (index < dialogue.Length)
        {
            textMesh.text = dialogue[index];
        }
        else
        {
            dialogueUI.SetActive(false);
            GetComponent<CursorLock>().LockCursor();
        }
    }



}
```

Appendix 32: DialogueTrigger.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DialogueTrigger : MonoBehaviour
{
    // Start is called before the first frame update
    public GameObject canvas;
    public static int size = 0;
    public string[] dialogue = new string[size];
    bool triggered = false;

    private void Start()
    {
        canvas = GameObject.Find("Canvas");
    }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject && !triggered)
        {
            canvas.GetComponent<Dialogue>().DisplayDialogue(dialogue);
            triggered = true;
        }
    }
}
```

Appendix 33: CursorLock.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CursorLock : MonoBehaviour
{
    public GameObject pauseMenu;
    public GameObject dialogueUI;
    public GameObject gameOverUI;

    public void ShowCursor()
    {
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
    }

    public void LockCursor()
    {
        if (!dialogueUI.activeSelf && !pauseMenu.activeSelf && !gameOverUI.activeSelf)
        {
            Cursor.lockState = CursorLockMode.Locked;
            Cursor.visible = false;
        }
    }
}
```

Appendix 34: ClearMenu.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class ClearMenu : MonoBehaviour
{
    public TextMeshProUGUI textmesh;


    private void Start()
    {
        Cursor.lockState = CursorLockMode.None;
        Cursor.visible = true;
        float clearTime = Time.time - GameManager.instance.startTime;
        if (clearTime < 10000)
        {
            textmesh.text = "Clear Time: " + clearTime.ToString("F1") + " second";
        }
        else
        {
            textmesh.text = "Clear Time: N/A";
        }

    }

    public void BackToMenu()
    {
        FadeUI.instance.StartFade("Menu");
    }
}
```

Appendix 35: NextLevel.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class NextLevel : MonoBehaviour
{
    public bool goal;
    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject.CompareTag("Player"))
        {
            GameManager.instance.currentHealth = Player.instance.state.currentHealth;
            if (goal)
            {
                FadeUI.instance.StartFade("Goal");
            }
            else
            {
                FadeUI.instance.StartFade("NextLevel");
            }
        }
    }
}
```

Appendix 36: BGMSwitcher.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class BGMSwitcher : MonoBehaviour
{
    public int index;
    public bool onStart = true;

    bool flag = true;



    void Start()
  {
        if (onStart)
        {
            PlayBGM();
        }
  }

    private void OnTriggerEnter(Collider other)
    {
        if (other.gameObject == Player.instance.gameObject && flag)
        {
            PlayBGM();
            flag = false;
        }
    }


    void PlayBGM()
    {
        if (index == 1)
        {
            GameManager.instance.playClearBGM();
        }
        else if (index == 0)
        {
            GameManager.instance.playNormalBGM();
        }
        else if (index == 2)
        {
            GameManager.instance.playBossBGM();
        }
    }
}
```

9. References

1. Gamedesigning, (2020, April 5), 10 Things Great Games Have in Common
   Retrieved from https://www.gamedesigning.org/gaming/great-games/
2. Unity Manual (2019, March), NavMeshAgent
   Retrieved from https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html