Maintenance Guide:

Table of Contents

1. Introduction:

This Guide will explain in detail about the below,

- Download and Installation of Unity hub,
- Download and Installation of Unity Editor
- Download and Installation of GitHub desktop
- How to clone the repository?
- How to setup the project from GitHub repo?

1.1. Unity Hub:

Navigate to the Unity website - https://unity.com/download

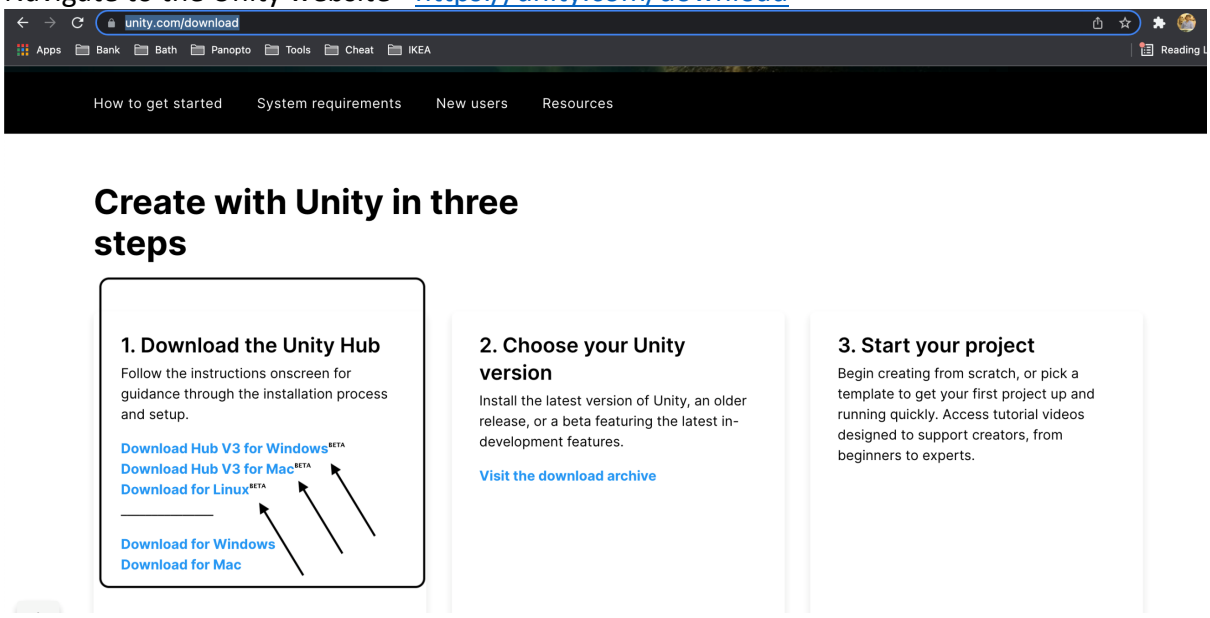

Figure1

Download the latest version depends on your operating system.
Once after downloading it, click on the executable file,

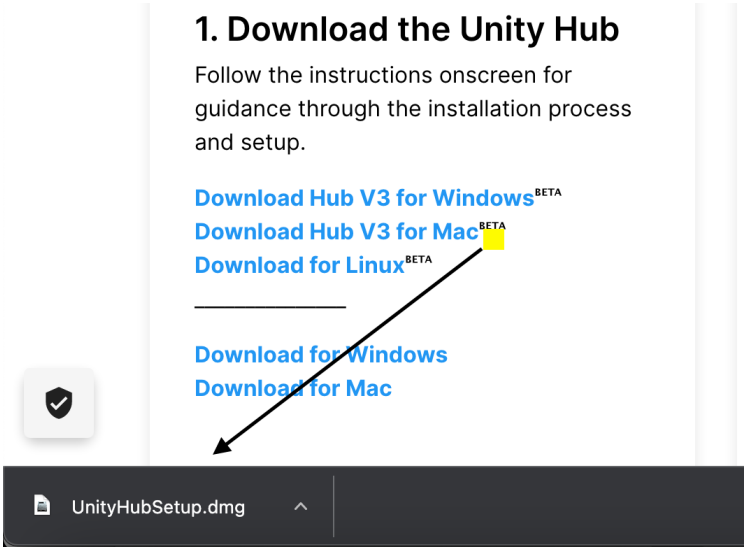

Figure2

this will open a window as below,

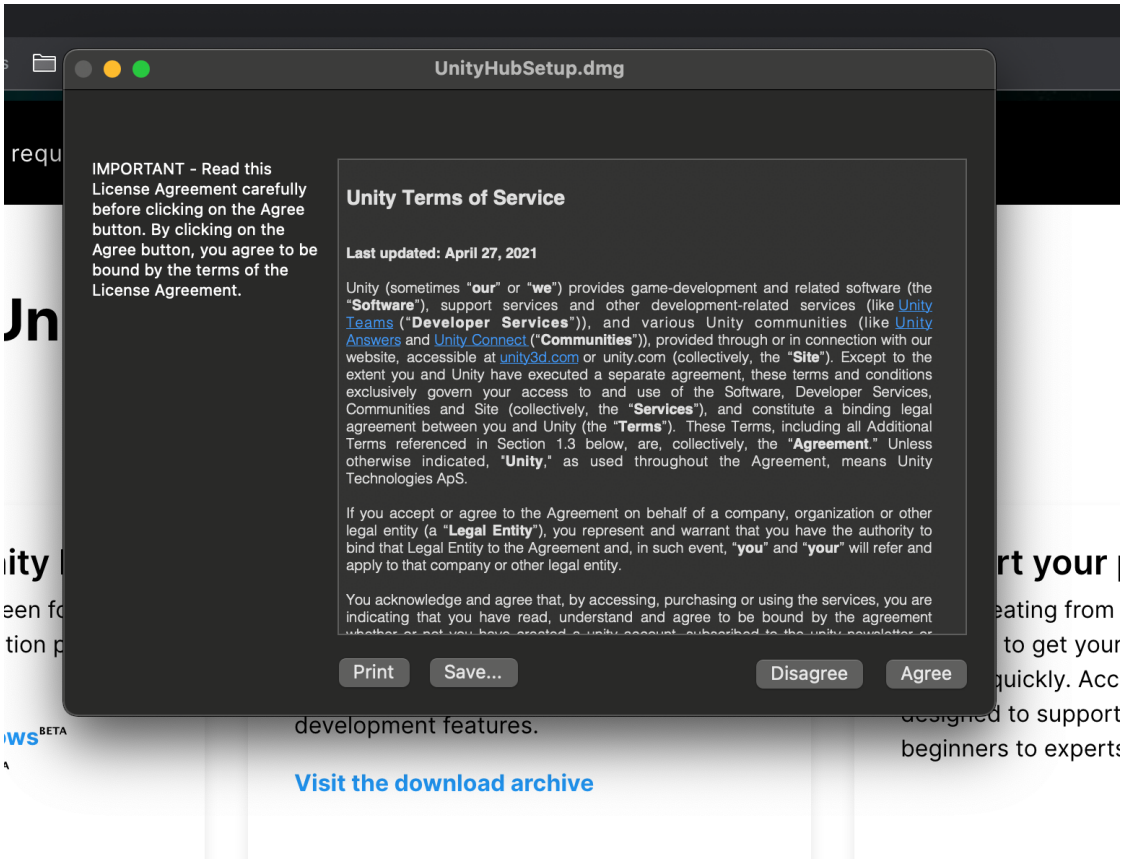

Figure3

Click on Agree and follow the onscreen steps for the installation.

1.2. Unity Editor:

Navigate to the Unity website - https://unity.com/download

And click on the step-2 Visit the download achieve

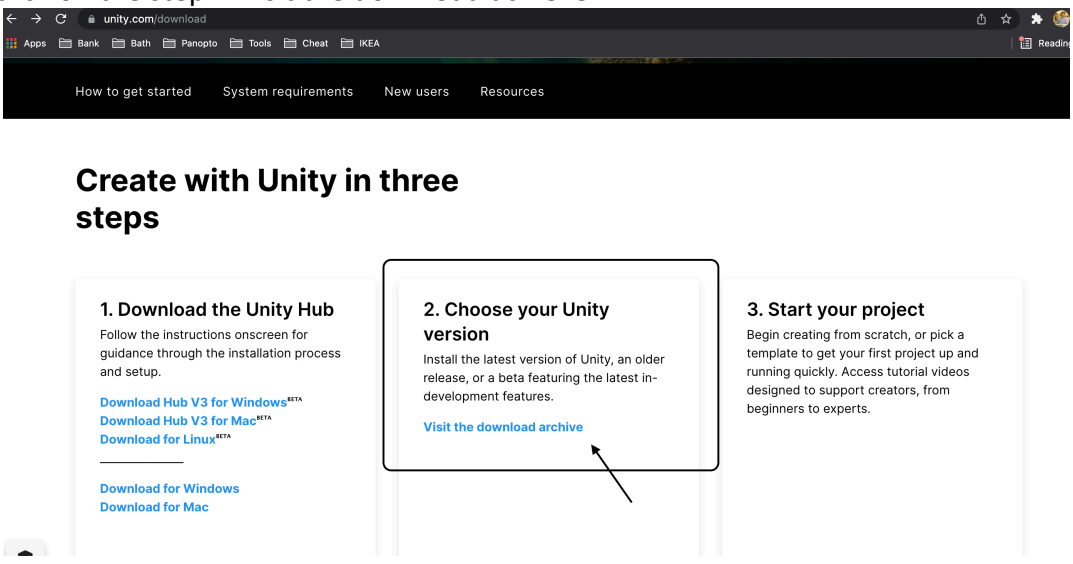

Figure4

This will open up the entire list of editors,

Our project runs on 2020.3.22f1, so download the corresponding version as below,

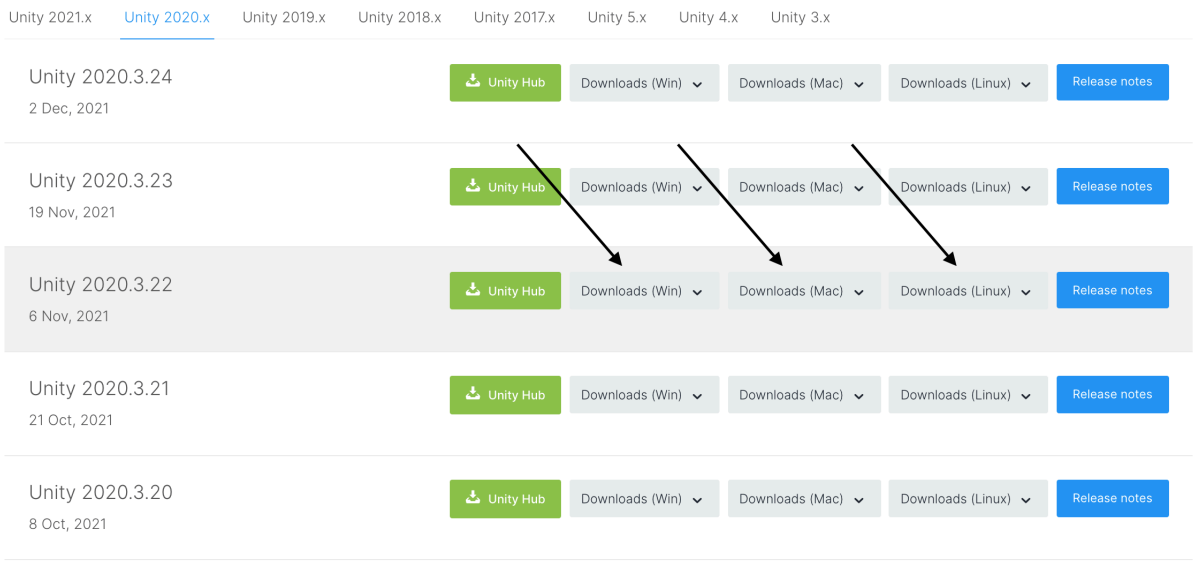

Figure5

1.3. GITHUB:

Navigate to https://desktop.github.com/

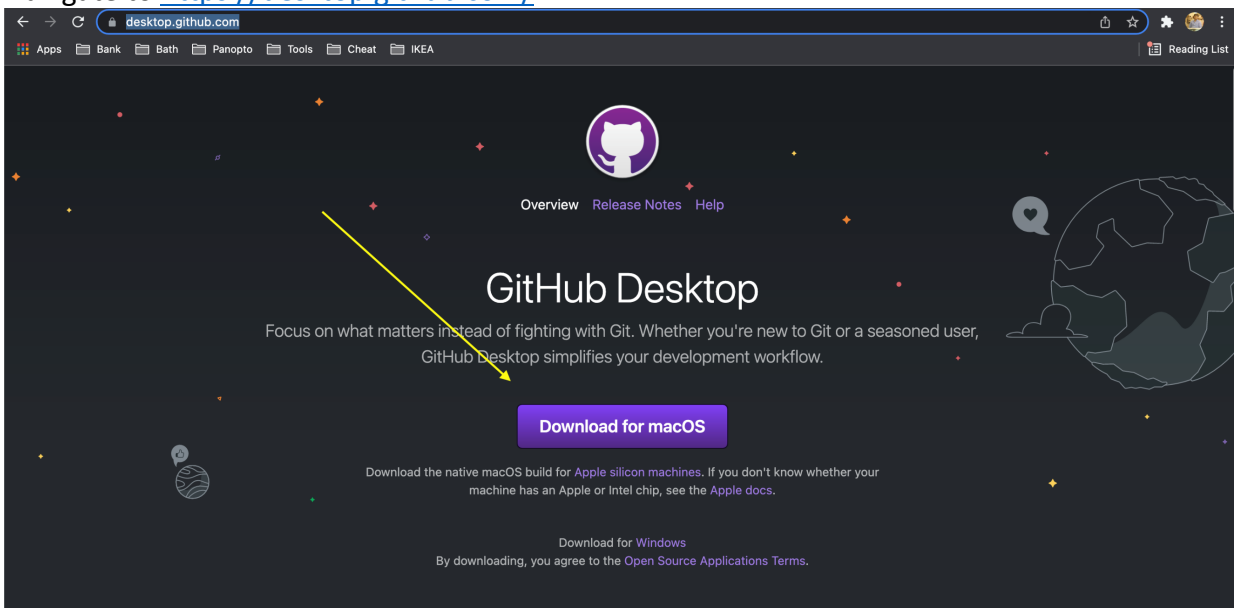

Figure6

Click on the Executable file and follow the onscreen process to install the GitHub desktop.

1.4. Clone Repository:

After installation, Open up the GITHUB client from your local machine,

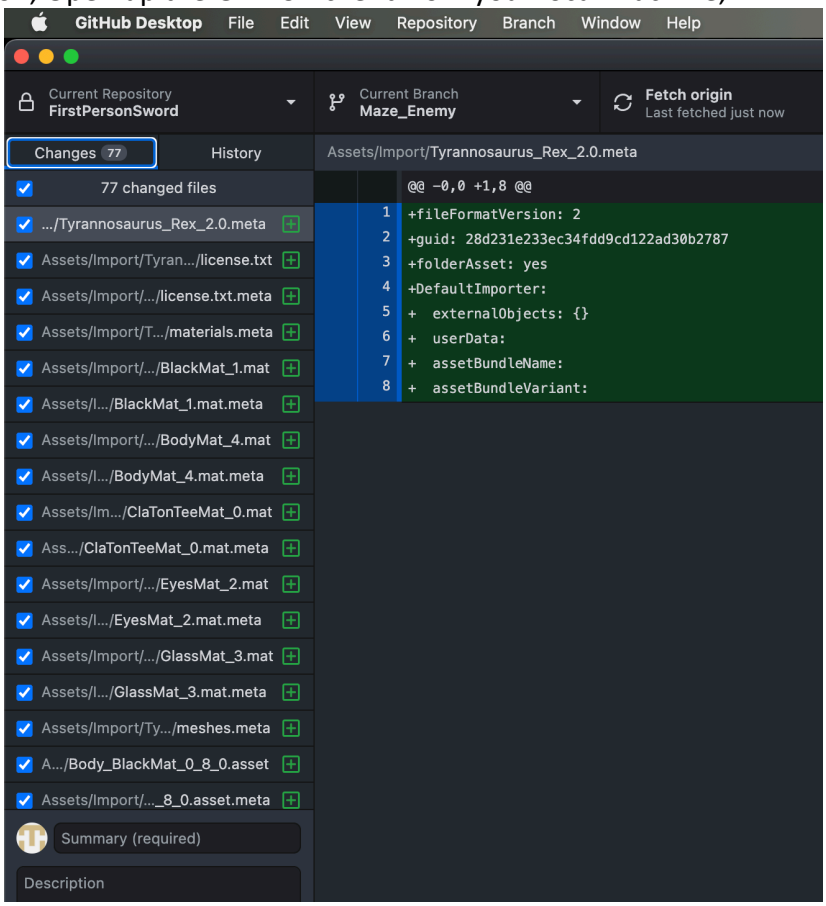

Figure7

From here, click on File → Clone Repository cmason2/FirstPersonSword

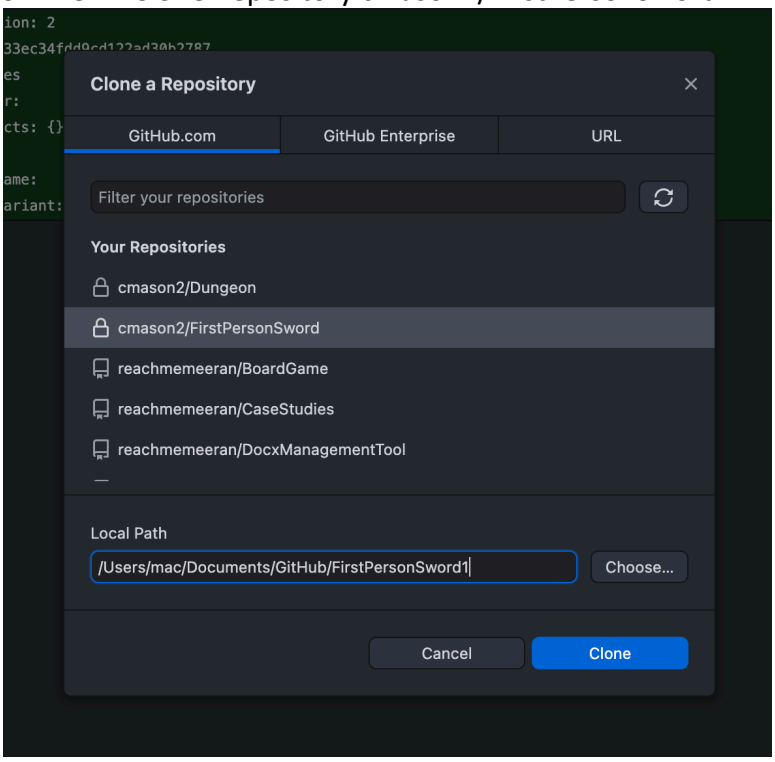

Figure8

1.5. Add a Project in Unity:

Open the unity hub client from your machine and click on Add

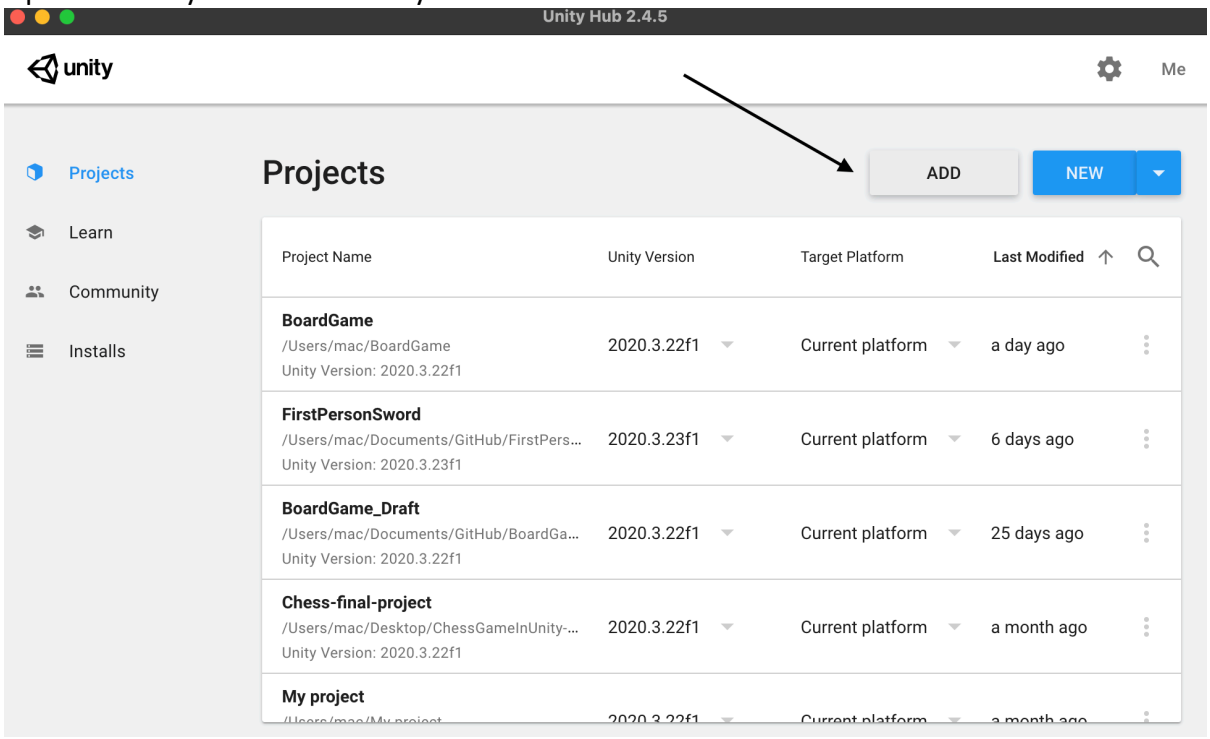

Figure9

and navigate to GitHub folder and select your project,

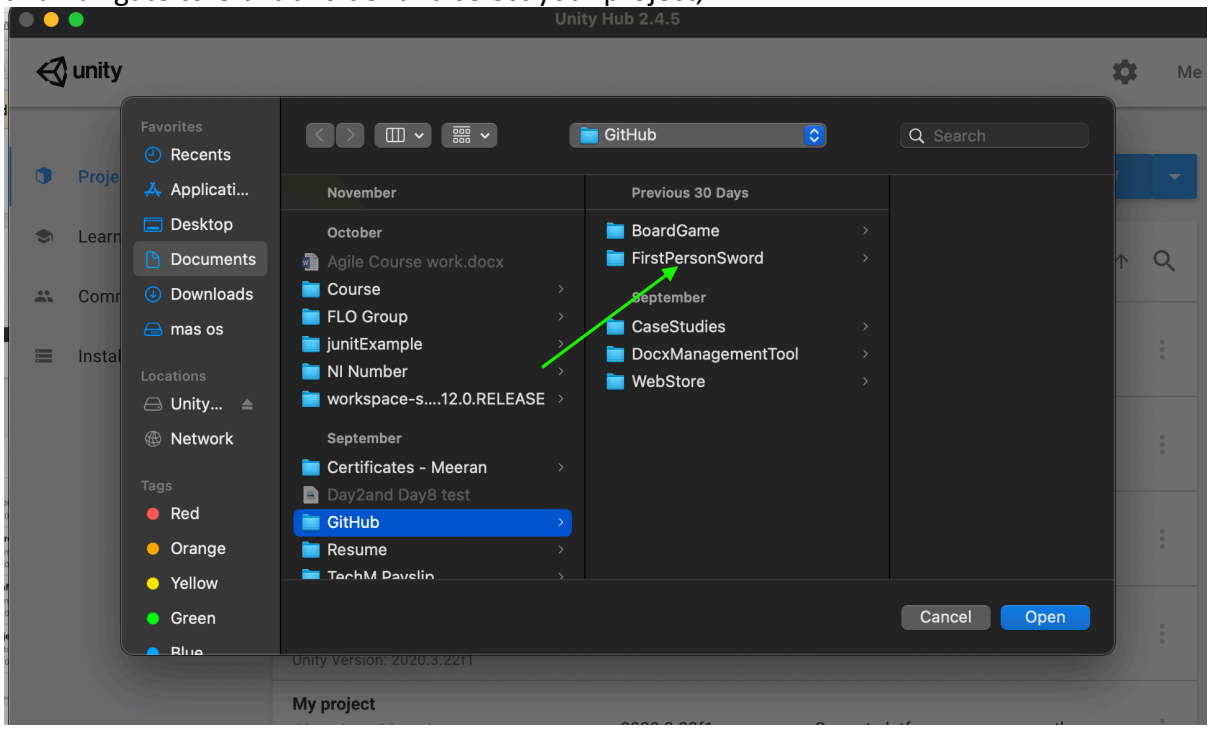

Figure10

You are now ready to work on this unity project to start developing the game.

2. Screenshots of the game:

Figure11: Menu Screen

The game starts of from here, the player can choose the level of difficulty as Easy, Medium or Hard. He can either quit the game from here or proceed to the first level.

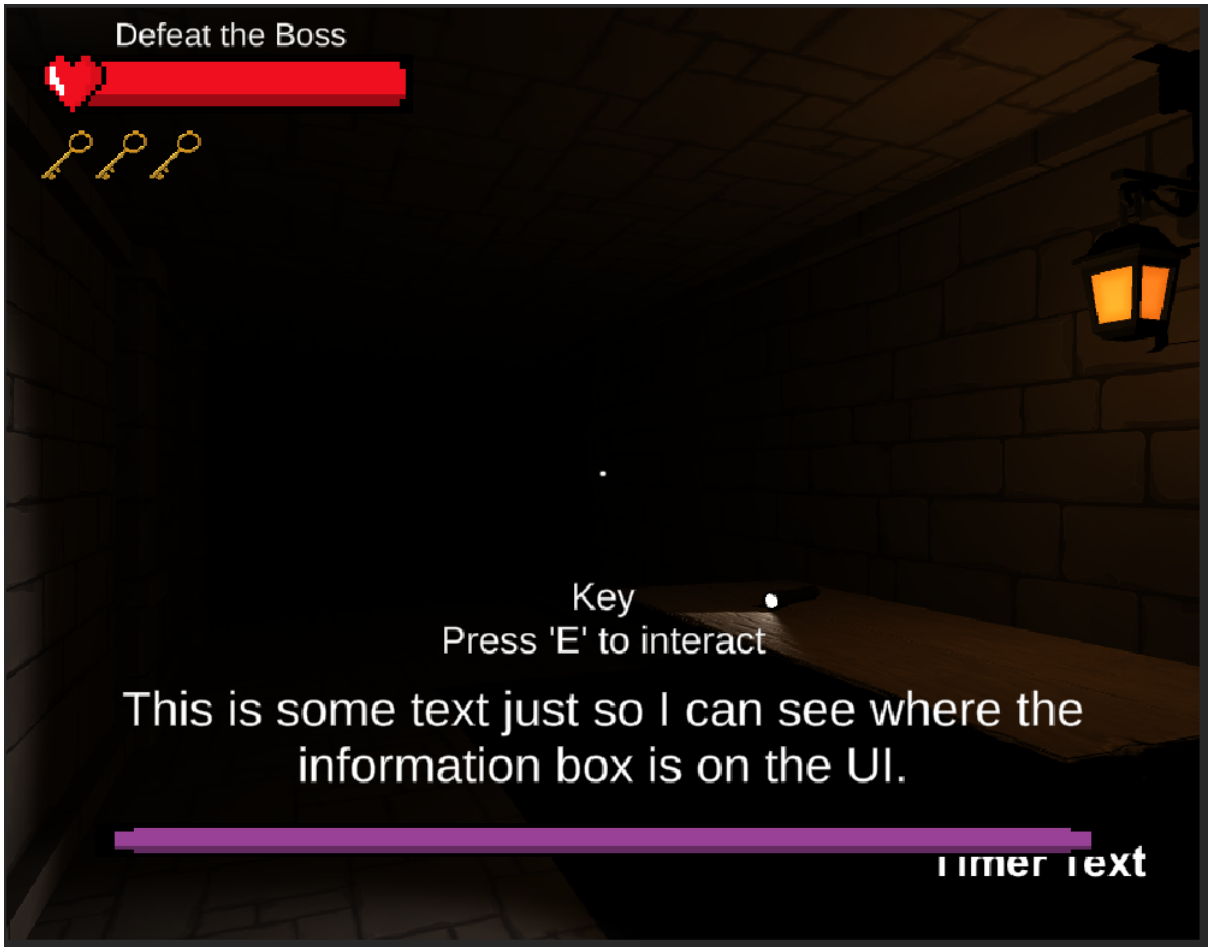

Figure12: Maze level

The player starts the first level in a maze. The maze level is designed to have limited vision. The player is expected to pick up a torch first and then goes onto find keys to open the door to next level.

Figure13: Spike level

The second level is the spike level, the player should kill monsters avoid unexpected spikes and finally reach the door to next level. Spikes can unexpected or static. Spikes can come from the floor and walls.

Figure14: Boss level

The boss level is the final level. The player has to defeat the big boss in the level to win the game. The boss is a monster which projects light at the player. The player has to deflect the projected light as well as kill other monsters in this level.

Figure15: Game Over Page

If player fails in any level, he comes to this page. Here the player has the option to go the main menu or retry the game once more.

3. Assets Used:

Figure16: Flying Bat

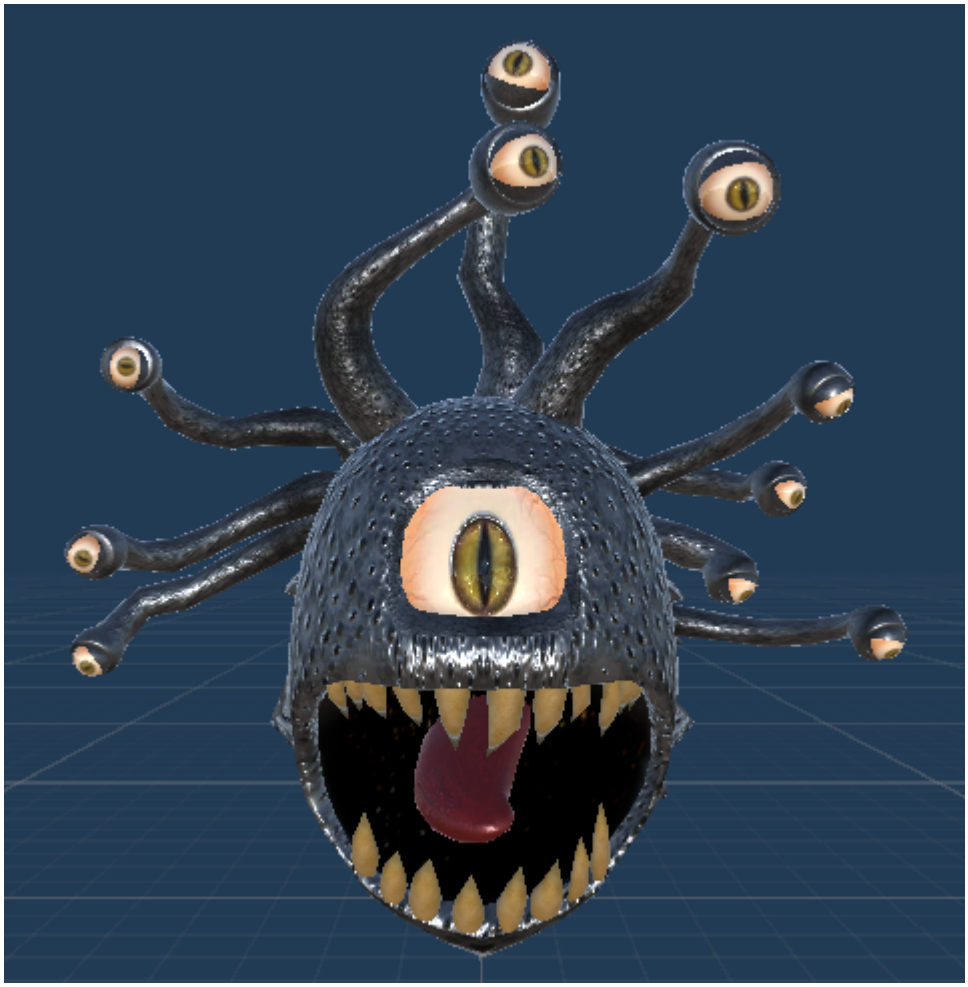

Figure17: Boss enemy

Figure18: The spider

Figure19: Barrels

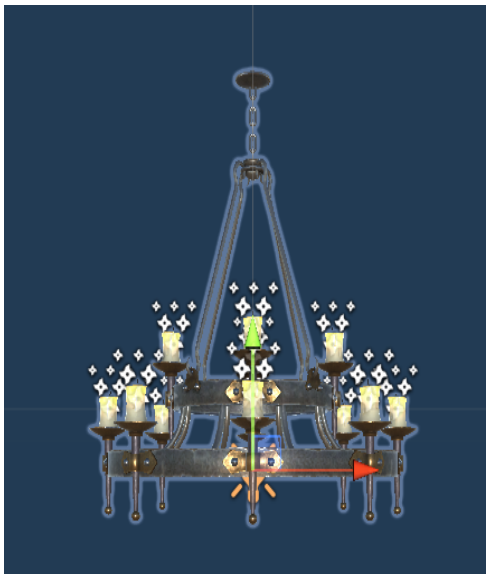

Figure20: Chandelier

Figure21: Wall lamp

Figure22: Medieval wall touch

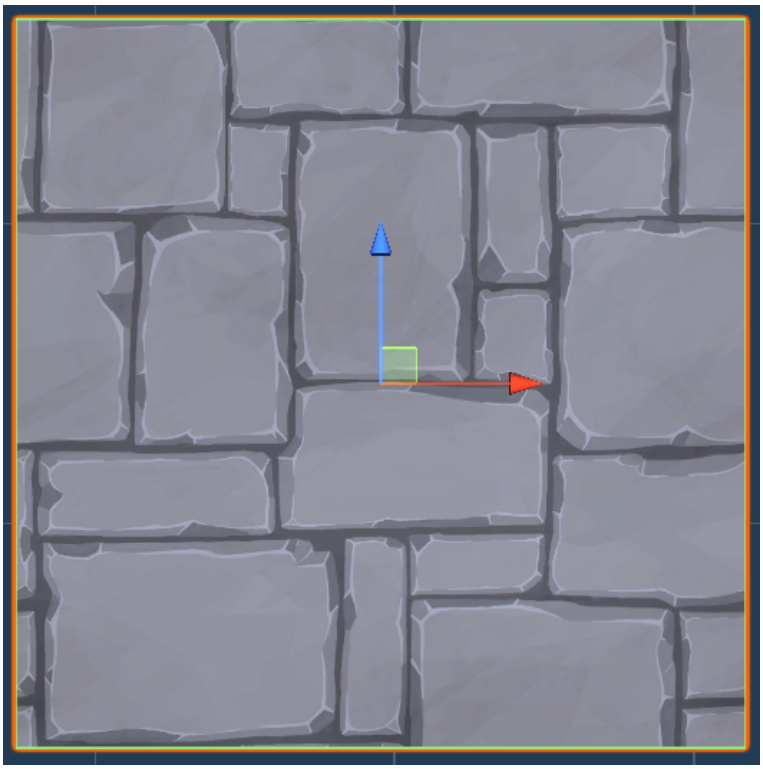

Figure23: Floor design

Figure 24: Medieval wooden chair

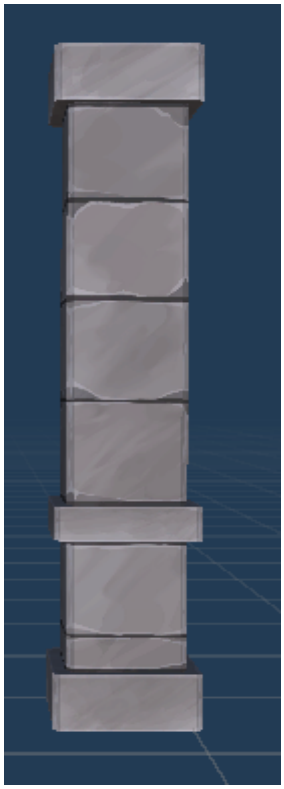

Figure 25: Pillar

Figure26: Wall Design

Figure 27: Wall Entrance

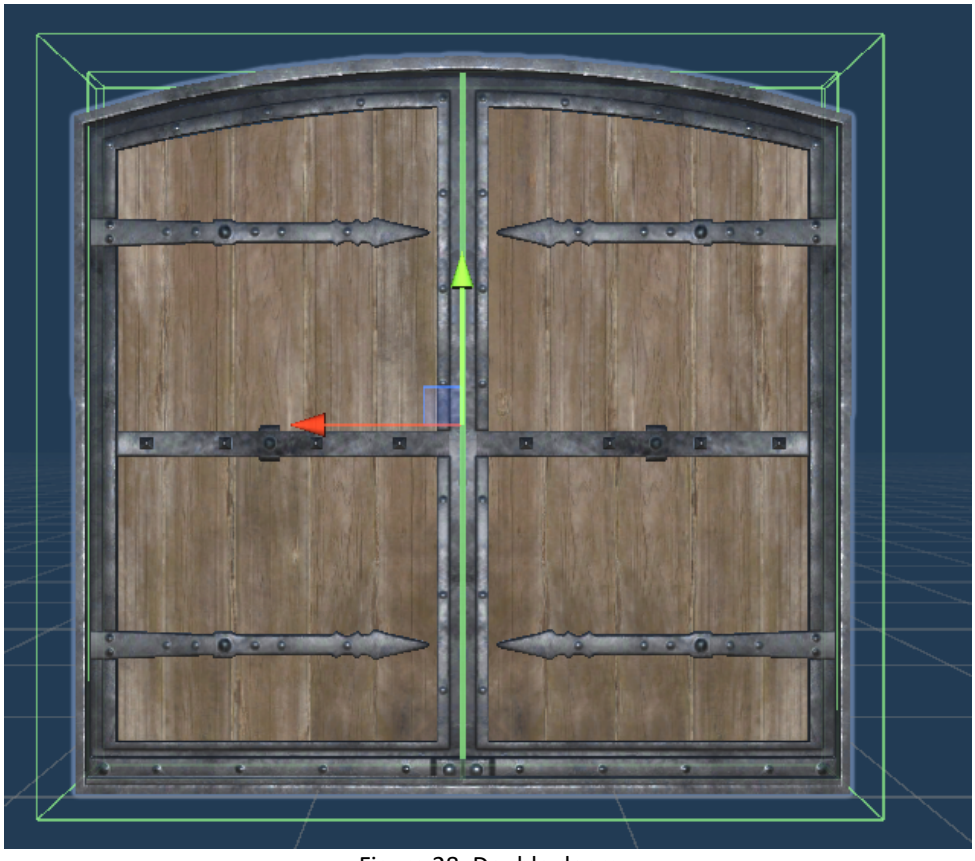

Figure 28: Double doors

Figure 29: Key

Figure 30: Player's Sword

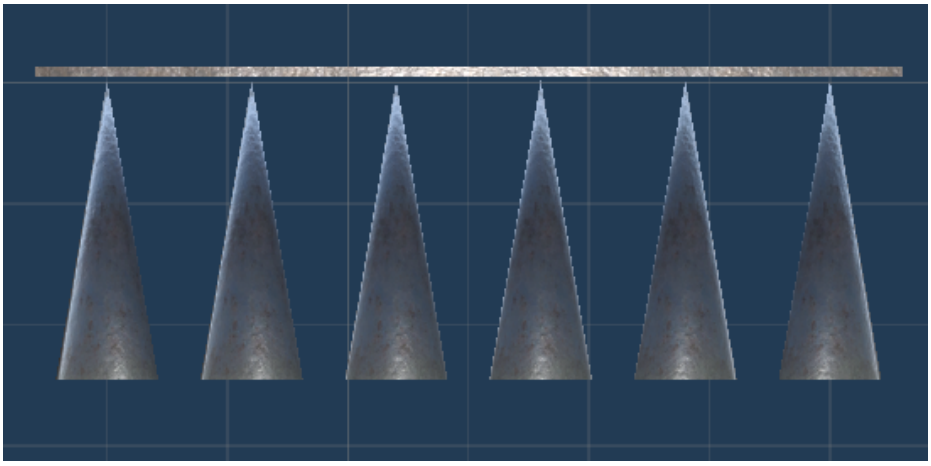

Figure 31: Wall Spikes

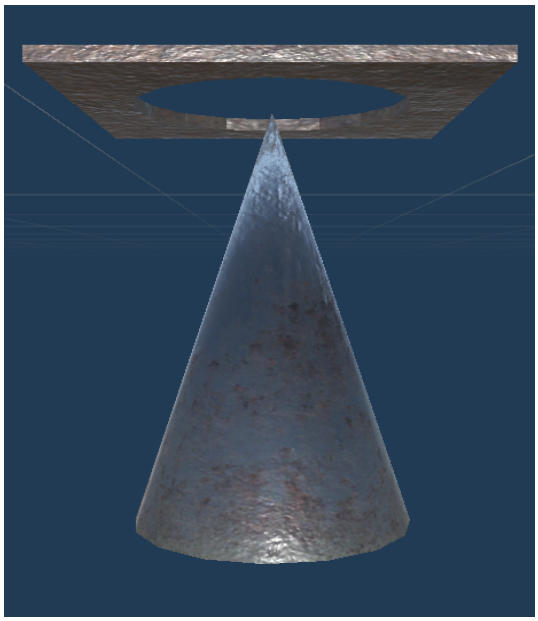

Figure 32: Spikes

Figure 33: Lever to stop the wall spikes

Figure 34: Double Door

Figure 35: Metal Cupboard

Figure 36: Interactable door

Figure 37: Health bar image

Figure 38: Number of keys collected image

Figure 39: Text showing the name of the game

Figure 40: Text showing the end of the game

3.1 Folders:

It is important to give attention to the folders when developing game. Game development is cumbersome process. It will be difficult to maintain and change things when the project is getting bigger. Organizing the folders at initial stage of the development makes it easier to change things later.

This is the main folders that developer should consider:

Import:

In this folder is where developers should put all asset that they download from outside sources. The mostly the downloaded asset are objects that developers can find from the internet such as Unity assets store.

Figure 41: Import folder

Material:

All colors and textures are saved inside this folder.

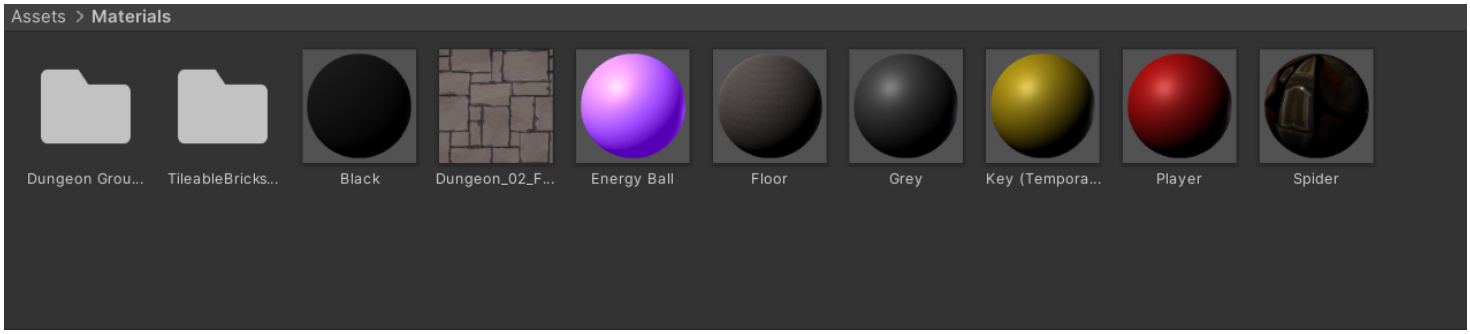

Figure 42: Material folder

Prefabs:

This folder is where all the objects that are reusable objects should put in. Reusable objects for Unity are objects that share pre-configured project hierarchies. This includes environment objects, interactable objects, and enemy.

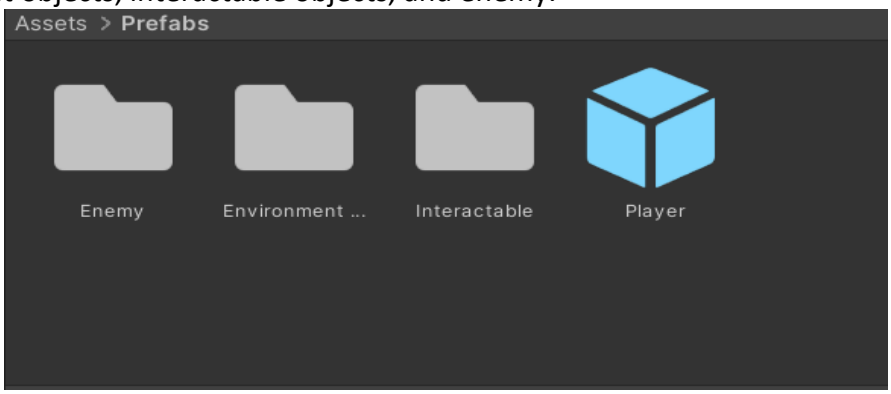

Figure 43: Prefabs folder

Environmental objects consist of wall, floor, flame, and other things that are used to create room or background.

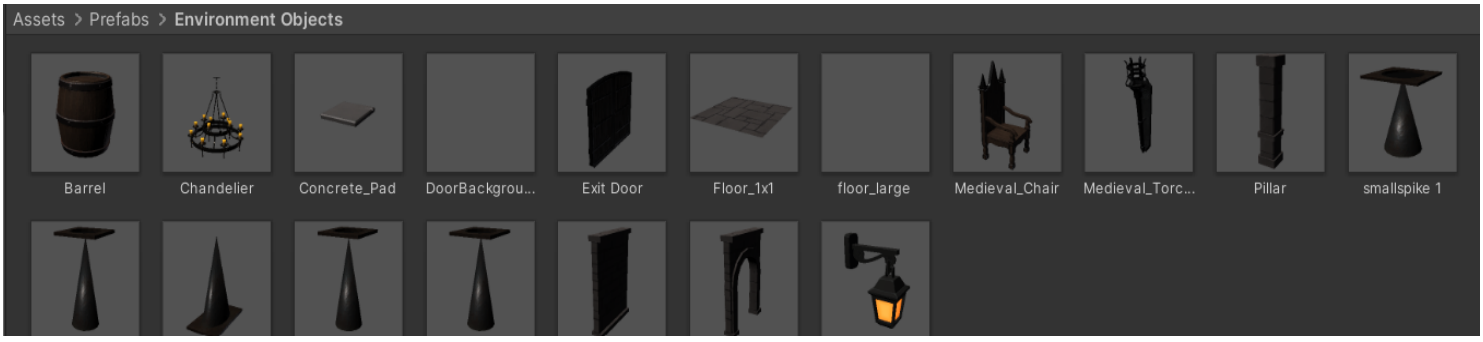

Figure 44: Environmental objects folder inside prefabs folder

Things that player can interact with such as doors, keys, flashlight, sword, etc. will group them inside the interactable objects folder.

Figure 45: Interactable objects folder inside prefabs folder

All monsters such as bats, spiders, and boss enemies are saved inside the enemy folder.

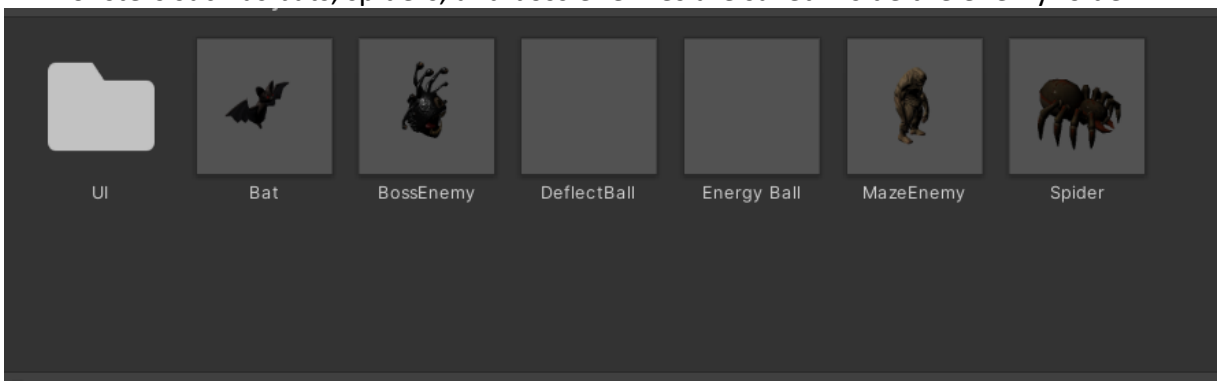

Figure 46: enemy folder inside prefabs folder

Scenes:

All scenes and levels that are created will be placed in this folder, which includes main menu, maze room, boss room, etc. When developers want to access each level, this folder is where you access it.

Figure 47: Scenes folder

Scripts:

This folder is where developer find all scripts (codes) that created. The Scripts folder includes enemies, player, interactable object, and UI scripts.

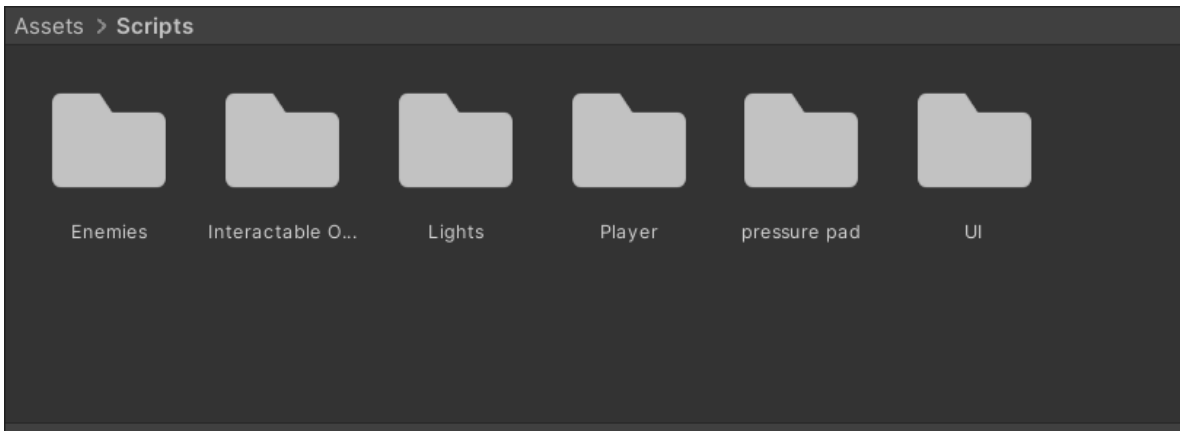

Figure 48: Scripts folder

UI:

all elements that are used to create user interface for this game saved in this folder. This includes main menu, text, item icons, etc.

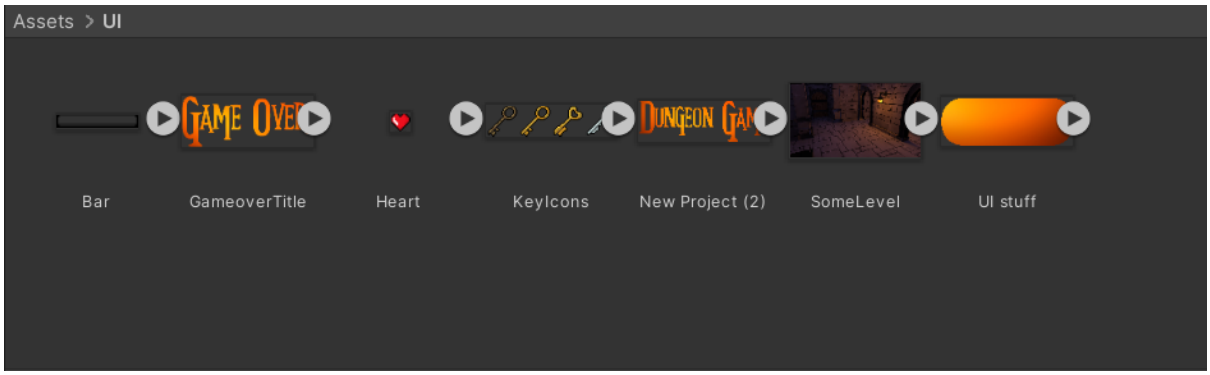

Figure 49: UI folder

3.2 Scripts:

The scripts are stored in the folder 'Scripts'. It is further divide for easy access.

Player:

The player folder contains the basic scripts for player element. It contains Player.cs and PlayerMovement.cs.

Player.cs:

(a) Start():
Function to set the initial requirements of the player like playerHealthBar, informationTextObject, infoText, interactText, sword_cam, timeSinceDamageTaken, timeSinceAttack etc. It also checks if the player is initially holding any equipment and calls EquipItem().

(b) Update():
This function updates the health of the enemy by checking if the damage time is greater than a minimum value. If this condition is true the enemy is considered to be vulnerable.

(c) AdjustCamera():
Function to rotate in the scene according to the movement of the mouse.

(d) CheckInteraction():
Whenever the key ‘E’ is pressed, by checking the target object appropriate animations are done. If the target object is doors, cupboards, chest etc, it is either opened or closed. If the object is sword or torch, it is picked up. If the object is keys, the count of the keys is changed. If the player is interacting with the final door of the level, the number of keys is counter before opening it.

(e) Attack():
This function is called when the player starts attacking the enemies and deflecting the rays from the boss enemy. Depending on the time taken by the player to swing the sword, the health of the enemy is reduced.

(f) Zoom():
Funciton that’s help the user zoom in and out.

(g) SetLayerRecursively():
The function is called when different layers need to be added to scene.

(h) CheckTorchStatus():
Function to turn the torch on and off.

(i) EquipItem():
This function is called whenever the player picks up equipment like torch or sword. These elements are then added to the first-person view.

(j) HoldItem():
This function enables the player to hold the equipment.

(k) DropItem():
Function to drop equipment, if anything is original picked up and when by the key ‘f' is pressed.

(l) CheckHealth():
This function checks the health of the player, that is it keeps checking if the player is dead or alive by comparing a threshold value ‘0’. If the value of the player’s health goes below zero he is considered dead.

(m) TakeDamage():
Each the player gets attacked by the enemies the player health reduces. This function updates the player health in the attacked scenarios.

(n) UpdateKeyUI():
Updates the keys count in the UI.

(o) SceneChangeNext():
Changes the scene the next immediate level when the player wins the current level.

(p) SceneChangeGameOver():
Changes from the current scene to the Game Over scene when the player loses the level.

The script file at the time of this document creation is below:


Player.cs

PlayerMovement.cs:

(a) MovePlayer():
This function controls the movement of the player in forward, left, right, backward and jumps.

The script file at the time of this document creation is below:


PlayerMovement.cs

UI:

This folder contains BossHealthBar.cs, GameOverUI.cs, HealthBar.cs, MenuUI.cs, SceneFade.cs, TextFadeInOut.cs and UITesting.cs

BossHealthBar.cs:

(a) Start():
The boss enemy's healthbar should only be activated if the scene is 'Boss level'. This health bar object is deleted in all other scenes.

(b) SetHealth():
This function sets the value of the boss healthbar.

(c) SetMaxHealth():
This function sets the maximum value for the boss healthbar.

The script file at the time of this document creation is below:


BossHealthBar.cs

GameOverUI.cs:

(a) startGame():
This function initializes the scene for the player. The first level is defined as the maze level.

(b) quitGame():
This function changes the scene to the menu scene when player quits the game.

The script file at the time of this document creation is below:


GameoverUI.cs

HealthBar.cs:

(a) SetHealth():
It sets the colour and value for health bar.

(b) SetMaxHealth():
It sets the maximum value for the health bar and the appropriate colour.

The script file at the time of this document creation is below:


HealthBar.cs

MenuUI.cs:

(a) startGame():
This function initializes the first scene for the player

(b) quitGame():
This function is activated when the player clicks on the Quit button. As the name suggests it exits the game.

(c) clickDifficulty():
This function is initializing the functions of the difficulty buttons: easy, medium and hard.

(d) leaderboardClick():
This function opens up the leaderboard scene to the player.

(e) easyDifficulty():
This function opens the game in easy mode.

(f) mediumDifficulty():
This function opens the game in medium difficulty mode.

(g) hardDifficulty():
This function opens the game in hard mode.

The script file at the time of this document creation is below:


MenuUI.cs

SceneFade.cs:

(a) Fade():
This function enables the fade in at the beginning of the scene and similarly at the fade out.

The script file at the time of this document creation is below:


SceneFade.cs

TextFadeInOut.cs:

(a) Start ():
Initialize InfoText variable with the component.

(b) DisplayTextFade():
Common function that enables text fades in scenes.

The script file at the time of this document creation is below:


TextFadeInOut.cs

UITesting.cs:

(a) Start ():
Sets all the button in game objects to false.

The script file at the time of this document creation is below:


UITesting.cs

Lights:

The Lights folder contains Flicker.cs

Flicker.cs:

(a) Start ():
Initializes the variables with components.

(b) Reset():
This function reset certains the values of MaxReduction, MaxIncrease, RateDamping, Strength variables.

(c) Update():
Updates the light status when necessary.

(d) DoFlicker():
Function to show flickering of the light component.

The script file at the time of this document creation is below:


Flicker.cs

Enemies:

The Enemies folder contains BatEnemy.cs, BatMovement.cs, BossEnemy.cs, BossMovement.cs, DeflectEnemy.cs, DeflectProjectile.cs, Enemy.cs, Projectile.cs, SpiderEnemy.cs and SpiderMovement.cs

BatEnemy.cs

(a) TakeDamage():
When enemy is attacked this function is called and its health is reduced.
(b) Death():
To destroy the game object.

The script file at the time of this document creation is below:


BatEnemy.cs

BatMovement.cs

(a) Start():
This function initialises variables with components.

(b) Update():
Called once per frame.

(c) SetDestination():
Function to set position for the enemy.

(d) DistanceToPlayer():
Calculate the distance to the player.

(e) Attack():
Attacks the player if the player is under the attacking distance.

(f) FaceTarget():
Function to keep facing the player.

(g) AttackCheck():
This function attacks the player when under the attacking distance.

The script file at the time of this document creation is below:


BatMovement.cs

BossEnemy.cs

(a) Start():
This function initialises variables with components.

(b) TakeDamage():
Reduces the health of the boss when he takes the attack.

(c) Death():
Destroys the game object if the boss have taken enough attacks.

(d) DeathAnimation():
This function animates the death of the boss enemy.

The script file at the time of this document creation is below:


BossEnemy.cs

BossMovement.cs

(a) Update():
Calls the FaceTarget().

(b) FaceTarget():
Face the player always.

(c) VerticalOscillation():
This function enables the vertical oscillation.

(d) ShootDeflect():
This function start shooting projectile

(e) SpawnEnemies():
This function enables the boss enemy to spew spiders and keep a count on them.

(f) HPInterruptedWait():
Checks if the health of the boss.

(g) MoveOverTime():
This function applies the transformation depending on start, end and time.

(h) Phase1():
In this phase the boss shoots projectile at the player and spawns spider enemies.

(i) Phase2():
In this phase, removes all other enemies from the level, deactivates pressure pads and retract bridge and shoot deflectable projectiles at the player.

(j) Start():
This function initialises variables with components.

The script file at the time of this document creation is below:


BossMovement.cs

DeflectEnemy.cs

The script file at the time of this document creation is below:

(a) Start():
This function initialises variables with components.

(b) TakeDamage():
Count the deflected projectile and change colours of the projectile as per a the count.

The script file at the time of this document creation is below:


DeflectEnemy.cs

DeflectProjectile.cs

(a) Start():
This function initialises variables with components.

(b) OnDestroy():
Destroys the projectile object.

(c) Update():

After fired the projectile should follow the player.

(d) FixedUpdate():
Increase the speed of the projectile.

(e) OnTriggerEnter():
Damages the player when collided by the boss, when collided with environment destroy the projectile and when deflected a number of times boss take the damage.

The script file at the time of this document creation is below:


DeflectProjectile.cs

Enemy.cs

(a) Start():
This function initialises variables with components.

(b) CheckHealth():
Check the health of the enemy when attacked by the player.

(c) TakeDamage():
Reduce the health of the enemy when attacked by the player.

(d) SetInvulnerability():
Depending on the vulnerability ApplyInvulnerabilityEffect() and ApplyInvulnerabilityEffect().

(e) ApplyInvulnerabilityEffect()
It applies the effect for invulnerability.

(f) RemoveInvulnerabilityEffect():
It removes the effect for invulnerability.

(g) ApplyDamageEffect():
Apply the effects of damage.

(h) Death():
Destroys the enemy objects when it has become vulnerable.

The script file at the time of this document creation is below:


Enemy.cs

Projectile.cs

(a) Start():
This function initialises variables with components.

(b) Update():
This function enables the projectile to moves in the direction of the player after being produced by the boss enemy.

(c) FixedUpdate():
This function enables the projectile to follow the player.

(d) OnTriggerEnter():
When the projectile collides with the player, it damages the player. When colliding with other environment objects the projectile is destroyed.

The script file at the time of this document creation is below:


Projectile.cs

SpiderEnemy.cs

(a) Start():
This function initialises variables with components.

(b) TakeDamage():
Function updates the health of the spider enemy whenever the enemy takes a damage.

(c) Death():
Destroys the object when the enemy is attacked beyond the threshold.

(d) DeathActions():
Destroys game objects.

The script file at the time of this document creation is below:


SpiderEnemy.cs

SpiderMovement.cs

(a) Start():
This function initialises certain variables with components.

(b) Update():
This function calls a set of other function per frame.

(c) UpdateAnimator():
Animate the spider in the forward direction.

(d) SetDestination():
Checks the distance to the player and moves the spiders position.

(e) DistanceToPlayer():
Calcuates the players distance from the agent.

(f) Attack():
The spider attacks the player if in a particular threshold distance called attackDistance.

(g) FaceTarget():
This function enables the spider to face the player always.

(h) AttackCheck():
Checks the player health to record the damage done to the player until player is vulnerable.

The script file at the time of this document creation is below:


SpiderMovement.cs

Interactable Objects:

The Interactable Objects folder contains the scripts for interacting objects. The scripts are Door.cs, Door1.cs.

Door.cs

(a) RotateDoor():
Function to rotate doors.

The script file at the time of this document creation is below:


Door.cs

Door1.cs

(a) Start():
Function initializes objects to animates.

(b) DoorOpen():
Function to open and close doors.

The script file at the time of this document creation is below:


Door1.cs

Spike:

This folder contains the scripts for the spikes and Lever folder. It contains Damage_smallspike.cs, Damage_traproom.cs, Spike_switch.cs and Spike_traproom.cs.

Damage_smallspike.cs

(a) Start():
Initializes variables to components.

(b) OnTriggerEnter():
Triggers the damage on the player until he is vulnerable.

The script file at the time of this document creation is below:

Damage_traproom.cs

(a) Start():
Initializes variables to certain objects.

(b) OnTriggerEnter():
Triggers the damage on the player until he is vulnerable.

The script file at the time of this document creation is below:

Spike_switch.cs

(a) Start():
Function initializes the component to a variables.

(b) OnTriggerEnter():
Triggers the animations for the objects

The script file at the time of this document creation is below:

Spike_traproom.cs.

(a) Start():
Function initializes component to a variable.

(b) OnTriggerEnter():

Triggers the animations.

The script file at the time of this document creation is below:


Spike_traproom.cs

Lever:

This folder contains Switch.cs

Switch.cs

(a) Start():
Initializes the spike object.

(b) MoveSpikesDown():
This function enables the downward movement of the spike.

The script file at the time of this document creation is below:


Switch.cs

Pressure Pads:

This folder contains Bridge.cs and Pad.cs

Bridge.cs

(a) Start():
This function initializes the variables for the script.

(b) bridgemoveforward():
This function is responsible for the forward movement of the bridge.

(c) bridgemovebackward():
This function is responsible for the backward movement of the bridge.

(d) CheckPadStatus():
This function checks the status of the pressure pads.

The script file at the time of this document creation is below:


Bridge.cs

Pad.cs

(a) Start():
This function initializes the object.

(b) OnTriggerEnter():
This function triggers the entry of the bridge.

(c) OnTriggerExit():
This function triggers the exit of the bridge.

The script file at the time of this document creation is below:


Pad.cs

4. Further Development

Even though the game is finished, there are still many features that should be developed and improved in the future.

In general, sound effect has not been added in the game yet. Every action that happens inside the game should have sound effect in order to make the game be more realistic. Also, pause menu that player can pause the game anytime he/she wants by clicking on the mouse has not been developed yet.

In term of the player, currently player can only move, pick the objects, and attack the enemies. There still have many rooms to improve the player's movement. For example, the player can sprint while running aways from enemies or the player can crouch when dodging enemies' attack. Moreover, player can only attack the enemy by using sword, which developer can add new attack pattern for the player such as player can use magic to shoot the fire from hand to enemies. In each level, player can have different weapons that makes game more entertain. For example, in spike level, the player has sword to attack the enemies while in the boss level, the player has bow to attack the enemies. Lastly, stamina bar could be developed for the player since the character that player play is human. The good game should consider the nature of human physical abilities.

In every level, it could have UI that can guide the player. The UI that gives the instructions of how the player should play in each level to have more interaction between player and the game. Adding the puzzle to find the keys in the maze room before moving to next level can make the game be more interesting and have variable game genres inside the game. For spike level, the layout of the level is too simple. The Developer can change the layout of the level to bring out the spike's feature. Creating complex room in certain degree makes the game more enjoyable. Then, there should have items can support the player while fighting the boss enemy in the boss level. It can be potion items that heal player health bar or shield that help player blocking the attack from enemies. In addition, final attack that defeat the boss enemy could improve to something else. For example, when the health of bar of boss enemy is almost empty, the player has to use lever that appears from the ground below the boss enemy to release the chandelier from ceiling and game is over.