MAINTENANCE GUIDE

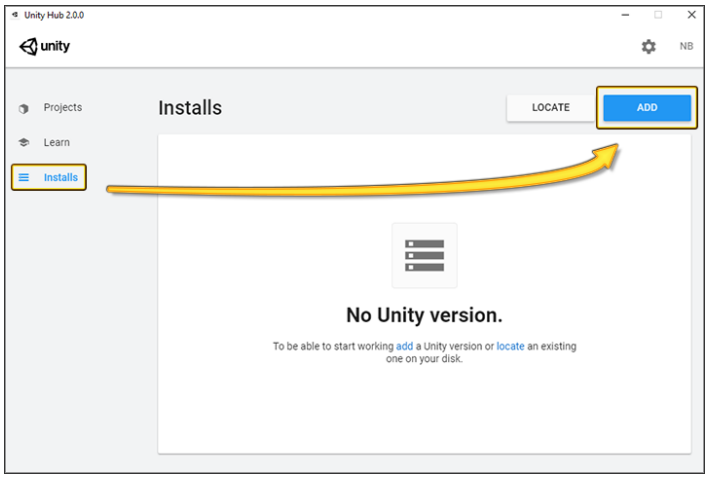
Contents

1. Developer installation steps: 2
2. Folder structure: 3
   a. Hierarchy 3
   b. Subfolder Description 4
   c. Autogenerated folders: Description 4
3. Class structure: 5
4. Unity Game Engine configuration: 5
   a. Game View: 5
   b. Project Folder Structure: 6
   c. Scenes: 6
   d. Scene Hierarchy: 6
   e. Inspector: 7
5. Unity Teams: 8
6. CRC CARDS 9

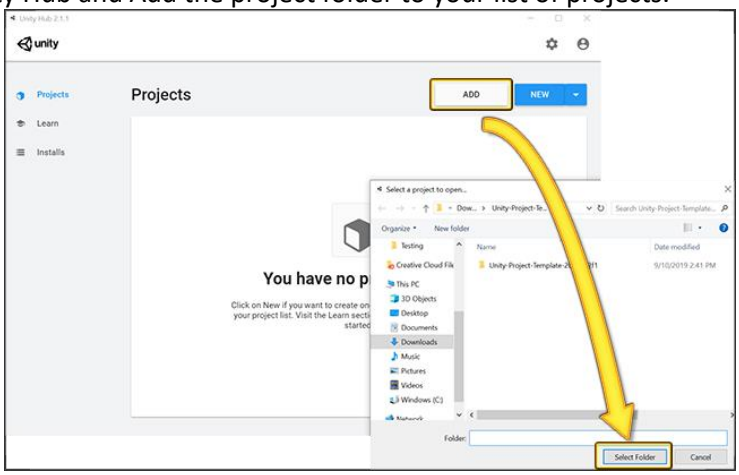
This guide describes the required steps for installing this project as a developer, the hierarchy of the project in terms of folder structure, class structure, and the project structure in Unity game engine, such that any developer with an interest in extending the game's functionalities and modifying the code can have an accessible reference for doing so.

1. Developer installation steps:

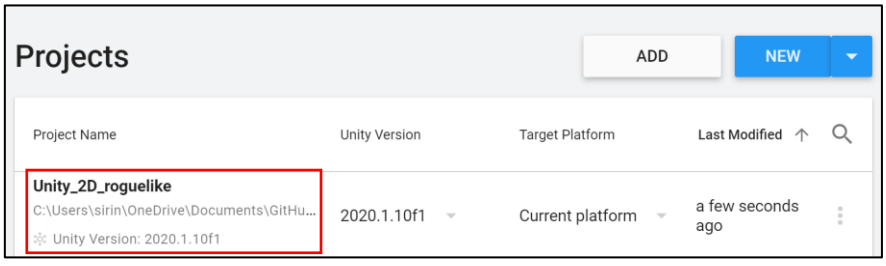
1. Download Unity Hub from the following link: https://unity3d.com/get-unity/download
   Follow the instructions on Unity Hub to download a version of Unity.
2. Add the downloaded Unity version 2020 by locating its file on your computer.



3. Install the Unity_2D_roguelike project from University of Bath GitHub by downloading a ZIP file of the project: https://github.bath.ac.uk/jpjf21/OliveGroup_SoftEngCW2
4. Unzip the project somewhere you remember on your computer.
5. Go back to Unity Hub and Add the project folder to your list of projects.



6. Click on the project in Unity Hub once it is loaded to open the project in Unity Game Engine.



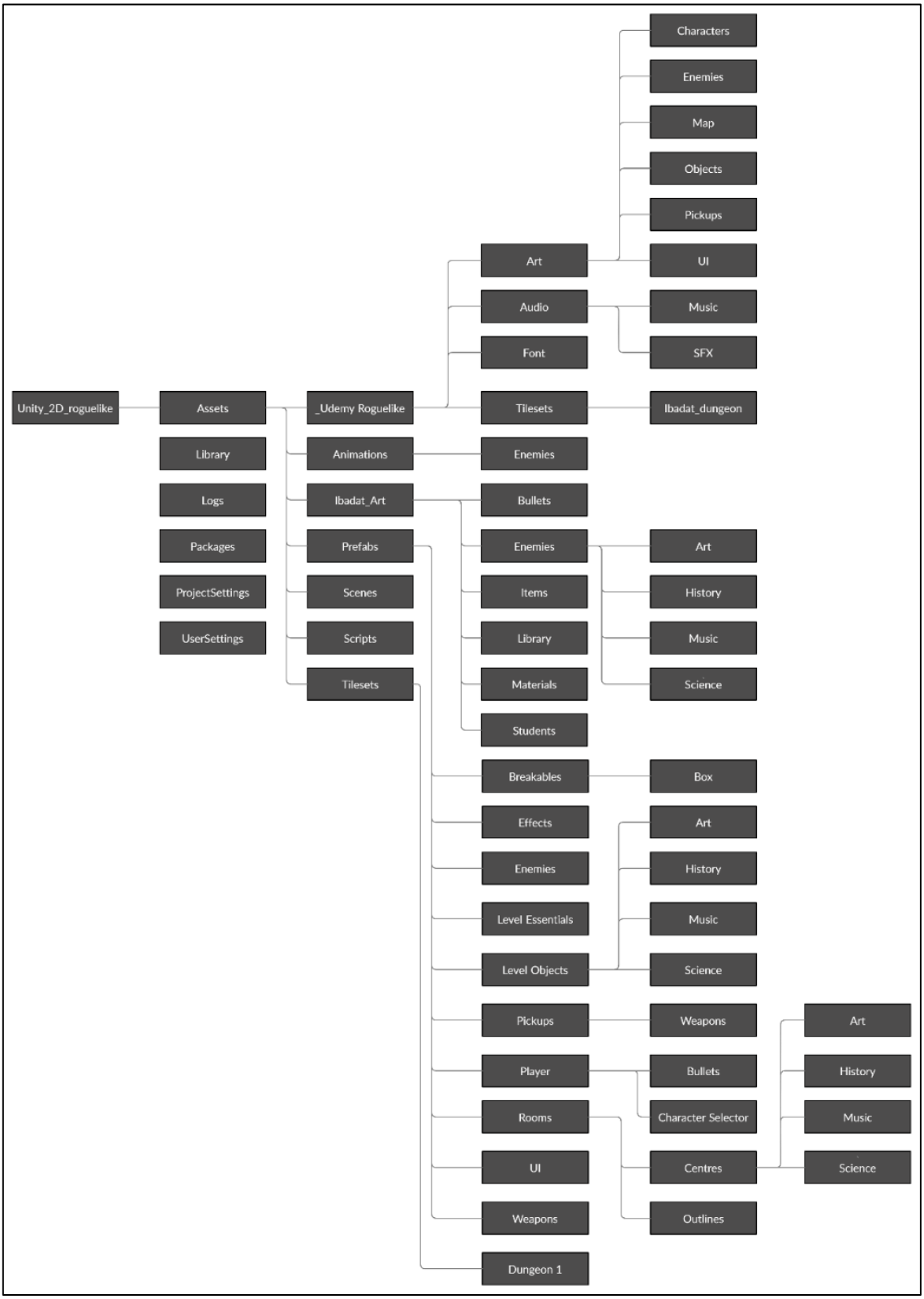
7. [Optional] Download Unity Teams for enabling collaboration among a team: https://unity.com/products/unity-teams

Now that you have the project open and you are ready to work on it in Unity, we shall explain the project structure and how to navigate between its folders and files, as well as a basic guide on how to navigate through the project in Unity Game Engine itself.

2. Folder structure:

a. Hierarchy

The hierarchy of the folders inside the Unity_2D_roguelike project folder is as follows:



(It is recommended to zoom in on this figure on your computer for clarity)

b. Subfolder Description

The project's subfolders on the first level are the following:

- **Assets:** This is the main folder where all development takes place. This folder contains all game objects that the developer needs to create the game, such as the art, the game scenes, and the scripts. The following are the sub-folders on the first level of the Assets folder as shown in more detail in figure:
  1. **_Udemy Roguelike:** This folder contains the art for the characters, enemies, map, objects, pickups, and UI, as well audio, font, and tilesets. It is the initial file location for this data, though we have also saved relevant art files in different locations for ease of access.
  2. **Animations:** This folder contains the animation data for the enemies in the game.
  3. **Ibadat_Art:** This folder contains the team's custom art for game elements such as bullets, enemies, items, students, and library objects. This is an extension of the _Udemy Roguelike > Art folder accessed from the same directory, and the developer can possibly use that as the single destination for all art data of the game.
  4. **Prefabs:** This folder contains the pre-configured reusable game objects in Unity which can be created in a scene and stored in the project. A game object is part of the Unity game engine, which generally gathers data configurations about specific objects found in a scene (this will be explained in more detail later). The game objects for this project are the folders found under Prefabs, such as Breakables, Effects, Enemies, etc.
  5. **Scenes:** This folder contains all Unity scene files in the game. Scenes will be described in more depth when we explain how the Unity game engine works.
  6. **Scripts:** This folder contains all Unity script files. Script files are the code segments associated with each scene, where a single script represents a class. These scripts will be documented in more detail in the following sections with an explanation on what each script/class does.
  7. **Tilesets:** This folder contains the tilesets for the dungeon rooms, which are used to design the levels themselves.

c. Autogenerated folders: Description

The following folders contain auto-generated data, so the developer does not have to focus on them while developing, though they can be useful for debugging compilation errors and retrieving past configurations:

- **Library:** This folder contains all cache files and meta data of the project.
- **Logs:** This folder contains the log files of the project.
- **Packages:** This folder contains the embedded packages used in the project, such as any specific libraries used for development. No external libraries are used in this specific project.
- **ProjectSettings:** This folder contains the files for the saved project settings (layout, collaboration mode, project version…).
- **UserSettings:** This folder contains the files for the saved settings for the user of Unity (subscribed with Unity).

3. Class structure:

The Scripts folder (Unity_2D_roguelike > Assets > Scripts) contains all the class files of the game, which are independent Visual C# Source Files that can be primarily opened in Microsoft Visual Studio IDE. Each script file (.cs file) represents a class and is associated to a single Game Object in the Unity engine, as we will see later. The classes are not dependent on each other; therefore, they are not formed in a hierarchy, and rather in a linear list of classes, each class being used in its associated Game Object.

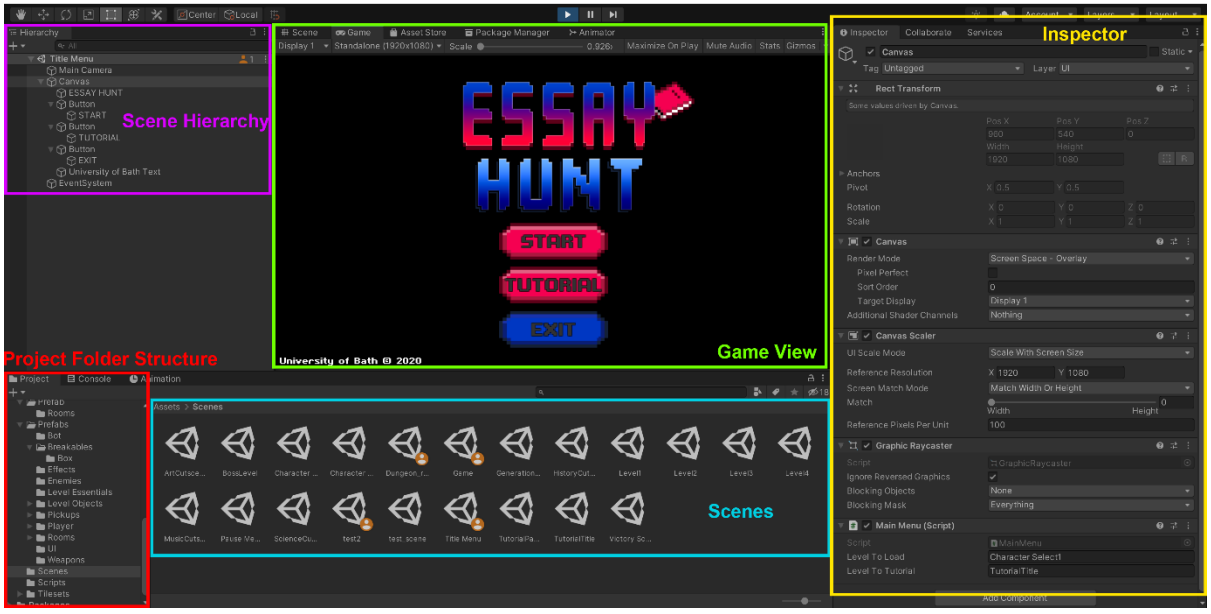
a. Class Structure Documentation

The list of classes with every class' description and attributes can be accessed from the project's automatically generated API documentation via Doxygen software "**Code Documentation.pdf**" file under the "**EssayHunt Documentation**" folder in the GitHub repository of the project:

https://github.bath.ac.uk/jpjf21/OliveGroup_SoftEngCW2/blob/master/EssayHunt%20Documentation/Code%20Documentation%20-%20Team%20Olive.pdf

4. Unity Game Engine configuration:

The following is a sample view of the main windows that appear upon opening the project in Unity:



For this specific setup, the windows are as follows:

a. Game View:

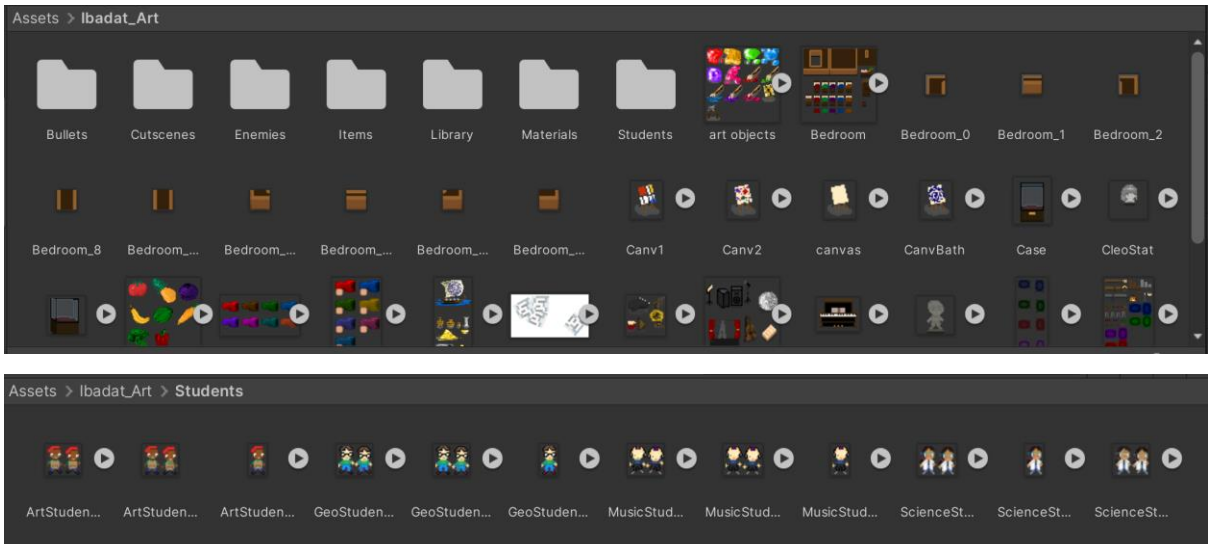
View of the game in play mode. This window also shows a Scene mode, where the game objects can be developed and modified directly through the interface. The developer can toggle between the Scene View and the Game View, where the development is done on the former, and testing the game is done on the latter. The following is an example of the "Title Menu" Scene in Scene View, where individual components (Game Objects) can be moved around and modified:

b. Project Folder Structure:

The exact same folder structure as explained previously, where the developer can open any folder from the structure, and the chosen folder appears for view in the window on the right. For example, the Scenes folder is chosen in the screenshot.

c. Scenes:

List of scenes in the game retrieved from “Assets > Scenes” folder in the folder structure, where the chosen scene is “Title Menu”. This view is not restricted to only the Scenes folder; this window can show any chosen folder from the Project Folder Structure on the left. For example, the following is a view of “Ibadat_Art” folder in the same window, which also shows this folder’s subfolders that can be directly selected and opened in the same window:

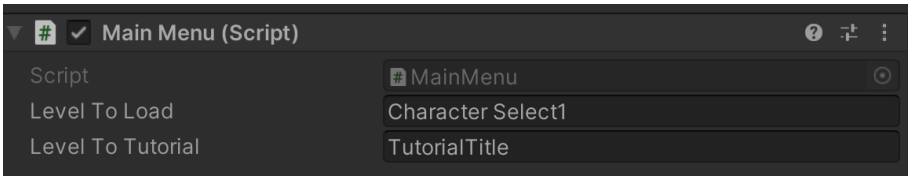


d. Scene Hierarchy:

This hierarchy contains the list of Game Objects in the current scene. For example, the “Title Menu” scene shown in the first figure contains the “ESSAY HUNT” Game Object which is the title logo of the game, as well as three buttons for Start, Tutorial, and Exit. Game Objects are scene components that can be configured using the Inspector window to generate behaviour that is defined by the programmer, such as colours and special effects. Game Objects can be combinations of different Game Objects; for example, in the screenshot shown, the Game Object “Canvas” combines Game Objects for the logo, the buttons, and the credit text.

e. Inspector:

This is the most important window in Unity Game Engine. It is where all Game Object configurations take place, and where scripts are linked to their specific Game Objects. Each Game Object has one associated Inspector View configuration. A Game Object can have an associated script, which is where the programmer writes instructions for the object's behaviour. The Inspector specifies other configurations that are not necessarily defined in the script, such as the number of rooms in a dungeon level, the colours of some UI elements such as buttons, and the sizes and fonts of the texts associated to them. In the shown screenshot, the Game Object "Canvas" combines all the Game Objects or elements on the "Title Menu" scene, and therefore can be programmed to generate behaviour for the whole scene. This is configured by clicking on "Canvas" in the Scene Hierarchy on the left, then opening its Inspector on the right, where the following script is associated to this specific Game Object, as shown again below for clarity:

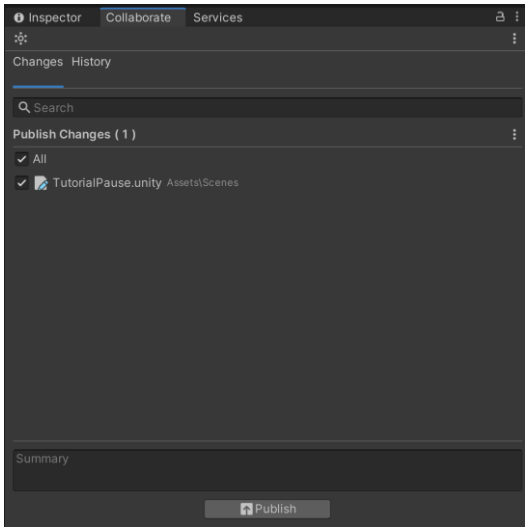


"MainMenu.cs" is the script associated to "Canvas" Game Object, which is shown in the initial screenshot. We can see other configurations such as "Level To Load", which is the name of the Scene that the current scene will link to upon clicking the Start button. The logic for this is of course coded in the script file, while the reference for the Scene itself is configured in the Inspector as shown. The same is done for "Level To Tutorial" configuration, where the Scene "TutorialTitle" for the tutorial menu is linked.

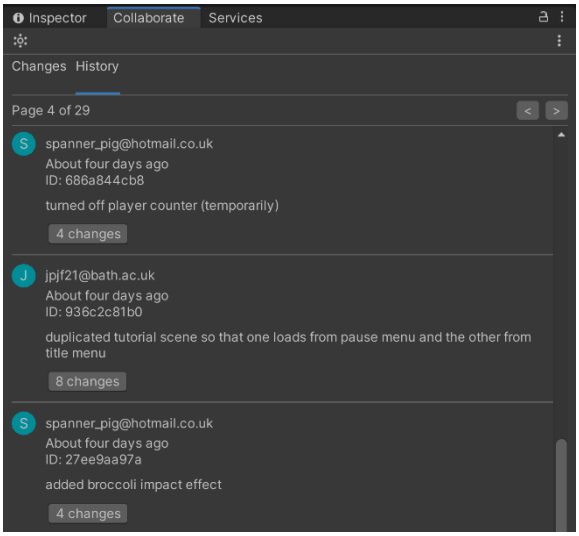
All of what is explained above can be modified to suit the developer's needs; for example, for the shown example of the "Title Menu" Scene, the colours of the buttons can be modified by clicking on one of the Button Game Objects in the Scene Hierarchy View, then changing the button's properties in the Inspector View. The same process is followed for all other Game Objects in any given Scene. **The most important thing to take from this is that for each Scene, there are Game Objects, and for each Game Object or collection of Game Objects, there is an Inspector configuration and an option for linking a script file.**

5. Unity Teams:

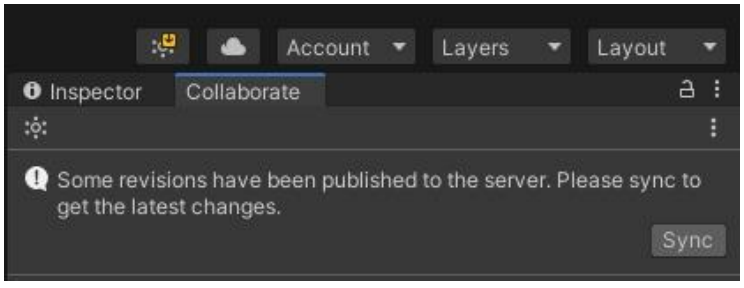
Unity Teams is a very efficient and quick collaboration tool designed for working on a Unity game project within a team. It is the development collaboration tool used in this project, and the following is a brief overview on using it. This requires Cloud Build to be installed, team members added, and the project saved to an organization within Unity Teams as explained in this tutorial: https://docs.unity3d.com/Manual/UnityCollaborateAddingTeammates.html. After those steps are set, the images on the following page show in the Collaborate tab next to the Inspector tab in the window on the right:



When changes are made, they can be published to the project on the cloud by adding a comment in Summary (optional) and clicking the “Publish” button.



History of all changes made by the entire team is shown in the History tab. Clicking on the button associated to an entry shows its detailed changes. You can also revert back to a particular set of changes in the event that something goes wrong.



If there are changes to the server (someone in the team has published a change), you can click on the "Sync" button to pull those changes into your own environment, thus updating your version of the project. Changes can be done simultaneously; teammates can publish (push) and sync (pull) changes at the same time without causing any conflict, even if they are working on the same files. You cannot sync until you have published new changes. You can choose to discard the changes to be published, as well as discard changes found in the server.

6. CRC CARDS

The following CRC Cards can be used alongside *Code Documentation – Team Olive.pdf* to make changes to the game.

breakables	
Responsibility	**Collaborators**
This class controls the breakable objects such as the chests.	MonoBehaviour

brokenPieces	
Responsibility	**Collaborators**
This class controls the broken pieces released by the breakable object.	MonoBehaviour

CameraController	
Responsibility	**Collaborators**
This class controls the camera.	MonoBehaviour

CharacterSelectManager	
Responsibility	**Collaborators**
This class controls the selection process of the game.	MonoBehaviour

CharacterSelector	
Responsibility	**Collaborators**
This class controls the character selection process in the game.	MonoBehaviour

DamagePlayer	
Responsibility	**Collaborators**
This class manages the player's health.	MonoBehaviour

EnemyBullet	
Responsibility	**Collaborators**
This class controls the bullets fired by the enemies.	MonoBehaviour

EnemyController	
Responsibility	**Collaborators**
This class controls the enemies.	MonoBehaviour

enemySleepAnimation	
Responsibility	**Collaborators**
This class controls the sleep animation of enemies when their health drops to zero.	MonoBehaviour

essayPageController	
Responsibility	**Collaborators**
Controller class for the essay page collectibles	MonoBehaviour

essayPages	
Responsibility	Collaborators
Class controls the collection of essay pages.	MonoBehaviour

Gun	
Responsibility	Collaborators
This class controls the guns used by the player.	MonoBehaviour

GunChest	
Responsibility	Collaborators
This class controls the chests where players can get new weapons.	MonoBehaviour

GunPickup	
Responsibility	Collaborators
This class enables the player pick up guns.	MonoBehaviour

healthPickup	
Responsibility	Collaborators
Class controls health pickups to restore player health	MonoBehaviour

LevelExit	
Responsibility	Collaborators
This class selects the next level to load.	MonoBehaviour

LevelGenerator	
Responsibility	Collaborators
This class generated levels randomly.	MonoBehaviour

LevelManager	
Responsibility	Collaborators
This class manages each level.	MonoBehaviour

LoadSceneAfterCutscene	
Responsibility	Collaborators
This class decides the scene to load after cutscenes.	MonoBehaviour

MainMenu	
Responsibility	Collaborators
This class creates and controls the main menu.	MonoBehaviour

PauseMenu	
Responsibility	Collaborators
This class to control pause menu scene.	MonoBehaviour

PlayerBullet	
Responsibility	Collaborators
This class controls bullets fired by the player.	MonoBehaviour

playerController	
Responsibility	Collaborators
This class controls the player.	MonoBehaviour

PlayerHealthController	
Responsibility	Collaborators
This class controls the health of the player and updates UI.	MonoBehaviour

RoomCentre	
Responsibility	Collaborators
This class controls the room centres.	MonoBehaviour

Room	
Responsibility	Collaborators

This class controls the operation of rooms.	MonoBehaviour

spriteSortOrder	
Responsibility	**Collaborators**
This class controls the ordering of the sprites.	MonoBehaviour

RoomPrefabs	
Responsibility	**Collaborators**
This class controls the room's prefabs.	MonoBehaviour

UIController	
Responsibility	**Collaborators**
This class is tasked with managing the User Interface.	MonoBehaviour

Tutorial	
Responsibility	**Collaborators**
This class controls the tutorial scene.	MonoBehaviour