PENG XIA

3D Game Development with Unity
A Case Study: A First-Person Shooter (FPS) Game

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

28 February 2014

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages	PENG XIA 3D Game Development with Unity A Case Study: A First-Person Shooter (FPS) Game 58 pages
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Markku Karhu, Head of Degree Programme in Information Technology

The project was carried out to develop a 3D game that could be used for fun and as an efficient way of teaching. The case of the project was a First-Person Shooting Game which the player had to search the enemies on a terrain map and kill the enemies by shooting. There were a few enemies on the terrain and they possessed four kinds of Artificial Intelligence (AI). The enemies would discover the player if the player entered into the range of patrol or shot them, and the enemies made long-range damage to the player. The player had to slay all enemies to win the game. If the player was slain, then the game was over. The Health Points (HPs) were used to judge whether the player or enemies died. The player could restore HPs according to touching the Heal Box or finishing a mission.

The game contains two main game scenes, one is the Graphic User Interfaces (GUI) scene and another one is the game scene. The player could play on or off the Background Music, view the information of the controller and the author and start or end this game on the GUI scene. The Skybox, Rigid body and Terrain were applied into the game. Also an interesting way of damage calculating was used in the game.

The game engine used for the project was Unity 4. Unity 4 was developed by the Unity Technologies. It is the synthesizing type of a game engine that the designers could use to develop a 3D video game, visualized constructions and real-time 3D animations. As well Unity is a cross-platform game engine, which means it supports the building of Windows OS, Mac, iOS, Android, Web, Xbox 360, PS3, Wii, Flash Player and Google Native Client.

The project demonstrates the basic features of the First-Personal Shooting Game and the process of the 3D game designing with the Unity 4 game engine. Also all assets and scripts are flexible for future development.

Keywords	3D Game, First-Person Shooter Game, Unity Game Engine, C#, .NET

Contents

1 Introduction

This project mainly deals with the development of a 3D game application with the Unity 4 game engine for Windows OS. Recently, the video game market appears to be of an unprecedented stage, which means the springing up of more platforms lead to more competition. The video game market is not just serviced for PC, PS3 and Xbox. The mobile platforms basis on iOS, Android and Windows Phone rise sharply. As a result, “cross-platform” come into people’s eyes.

Real time 3D games have existed for approximately ten years now. We have played them, created assets in the style of our favorites, and maybe even “moded” a few of them. However until recently, the cost of licensing one of the premier game engines has ranged from several hundred thousand to several million dollars per title, relegating the dream of creating one’s own 3D game to an unattainable fantasy. With Unity’s bold move to offer a robustly featured free version of their engine, a radical change in the pricing models of the high-end engines has rocked the industry which be willing to take high cost to make games or CG (Computer Graphics).Unity 3D game engine is the most professional, steady and efficient game engine, and Unity 3D game engine supports Web, PC, Mac, iOS, Flash, Android, Xbox360, PS3 and Wii platforms. [1, 3] The project used the Unity 3D game engine to develop a 3D game, which the case of the game is a first person shooter (FPS) game.

The report is aimed at people who are beginning with learning Unity and possess at least a basic knowledge of the Unity 3D game engine. It does not discuss the whole process of creating the game. Chapter 2 discusses the theories of the game design, chapter 3 introduces and demonstrates the Unity game engine, chapter 4 discusses the implementation of the project, and chapter 5 discusses the future development and the feasibility of the market. The project is not developed with ways of generating revenue. Using the game for recreating and teaching purposes is more suitable for it.

2 3D Game Design Theories

2.1 2D/3D Theories

An understanding of a motion and the driving forces thereof is crucial in understanding games. Most game objects in games move. What makes them dynamic is that. If it is a 2D character such as the premier Angry Birds or a fully-fledged 3D character such as Tomb Raider, their game environments and they are in constant motion. To comprehend the concept of motion, particularly with respect to computer games, a creation of foundation knowledge in vector mathematics is required. The vectors are used comprehensively in game development for describing not only speed, acceleration, position and direction but also within 3D models to specify UV texturing, lighting properties, and other special effects. [1,175.]

2.1.1 Principles of Vectors

In 2D, a vector has x and y coordinates. But in 3D, it has x, y, and z coordinates. In unalloyed mathematic system, a vector is not only a point in plane, but also a set of dynamic coordinate instructions. There is a legend to be shown for understanding the vectors. A fabled treasure map is created, and a cargo ship is stopping at the point (4.8) called start. For an example, the sailors take three steps to the east and seven steps to the south. As shown in Figure 1, the sailors move three steps to the east and a vector (3, 0) is represented, which means move 3 in a positive x direction. Then the sailors take seven steps to the south could be represented that a vector (0, -7), which means move 7 in a negative y direction and 0 in the y direction. [1,175.]

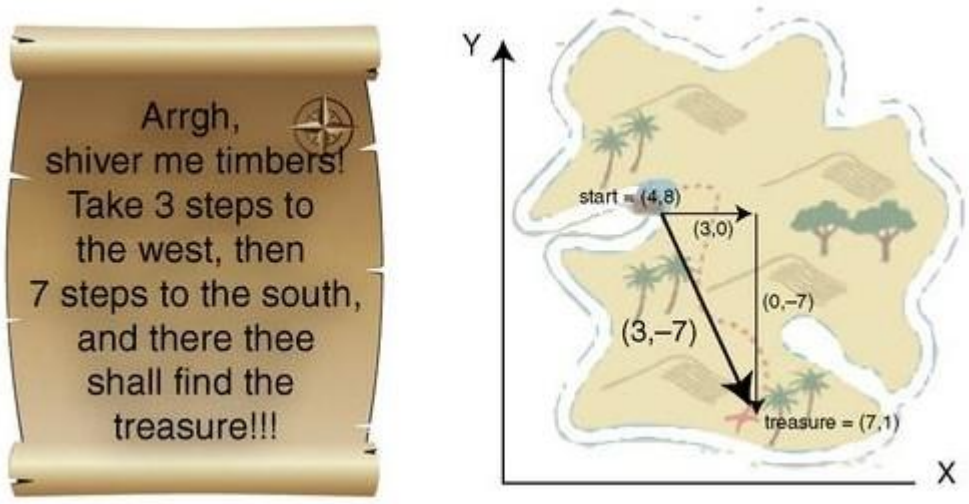


Figure 1. An imaginative map illustrating the usage of vectors. [1,176.]

To determine the final location, vector x and y values are added to the starting point x and y values. For example, in Figure 1, the cargo ship lands at the point(4, 8) and the sailors move positive three to x-axis, which will place them at (7, 8). Then they move minus seven to y-axis, which will place them at (7, 1). Actually, they can also arrive the same point by taking a shortcut, in other words, they can walk directly in a straight line between the start point and the treasure. In the case, the two vectors (3, 0) and (0, −7) must be combined together and the result is (3, −7). By using this new vector and departing from the starting location, they will reach the same location at point (7, 1).

To come back to the cargo ship from the treasure, the sailors can walk along the same paths which were illustrated above but in the reverse direction. This can be completed by reversing the vector so that all coordinate values should be multiplied by − 1. In the case, in order to get back to the ship they have to go along the vector (− 3, 7). It is the best way for the persons to know the straight distance between the ship and the treasure. The length of a vector, the representative is v, defined its magnitude and written |v|, and it can be found by using Pythagoras' theorem, as shown in equation (1).

$$|v| = \sqrt{v \cdot x^2 + v \cdot y^2} \quad (1)$$

Every game object or prefab in Unity has plenty of vectors which are related to it. The transform component of a game object or prefab has three important properties: rotation, position, and scale. The layouts of a classic game environment with the character

models as the objects are shown in Figure 2. Generally, in 3D, the x axis represents to the side, the y axis up, and the z axis forward. Every game object or prefab has its own transform property. The axes are indicated in the Scene window with three kinds of arrowed lines.as red, which is shown in Figure 2, they are coloured as red, green and blue. The y axis is green, the x axis is red, and the z axis is blue.

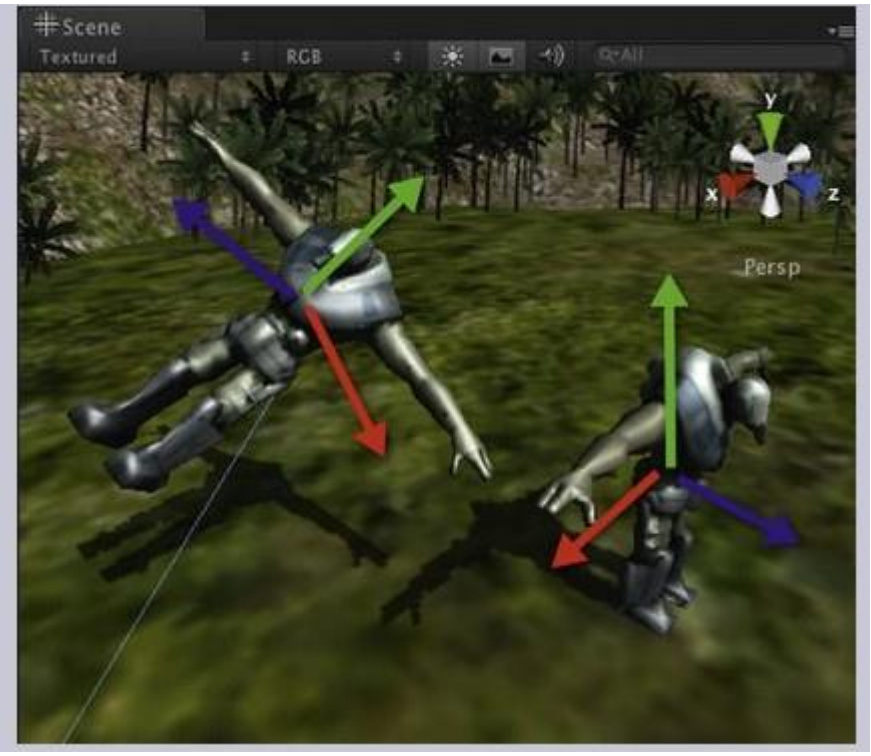


Figure 2. The coordinates in the Edit Scene of Unity.

The environment in Unity has its own axes, and the orientation can be adjusted by the approach which the user changes the viewing angle around to observe different objects. In Unity, the orientation of the camera determines the main orientation. Also all game objects have their own orientations drawn by x, y, z axes showing in the Scene window while the game object is pitched on. Thus, by lying down a game object, the y axis of this game object could be horizontal locally.

In Unity, the Vector2 and Vector3 classes are generally used. The position, rotation, and scale values of a game object are stored as Vector3. The 2D vector information is stored as Vector 2. a vector of a game object for x, y and z axes can be controled by programming the codes Vector3.left, Vector3.up, and Vector3.forward respectively. The users can move a game object along its axes without any code and its orientation.

2.1.2 Defining 2D and 3D Space

The theories of vectors in 2D or 3D space are applied in the same way. The difference between a 2D coordinate system and a 3D coordinate system is the value of another parameter. In 3D game engines, 2D games are developed by x and z axes. In 2D games, generally every game object is positioned in the same plane, which has an initialized value of y axis defined to be 0. Any movement of the game object thereafter only can act in the plane. It is can be assimilated with moving an object around on a planar ground top. In most 2D games, the main camera is placed directly above the scene and perspective is removed to give the view of a truly 2D scene. Whether it is in 2D or 3D game, the camera is a critical component because it displays all actions of the game objects to the player on the game scene. As a result the lens through the game scene is perceived. Comprehending how the camera moves and how to adjust what it acquires is essential knowledge. [1,185.]

Bi-dimensional (Two-dimensional) space is a geometric model of the planar projection of the physical universe in which we live. The two dimensions are commonly called length and width. Both directions lie in the same plane. To analyse a 2D case, as shown in Figure 3, that of the classic Flatland example, in which a person lives in a 2D universe and is only aware of two dimensions (shown as the blue grid), or plane, say in the x and y direction. Such a person can never conceive the meaning of height in the z direction, thus he cannot look up or down, and can see other 2D persons as shapes on the flat surface he or she lives in.

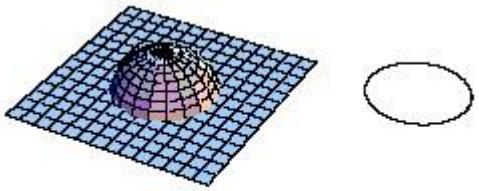


Figure 3. Understanding 2D space lab. Copied from Engineer Xavier Borg (2007) [2.]

Three-dimensional space is a geometric three-parameter model of the physical universe (without considering time) in which all known matter exists. These three dimensions can be labelled by a combination of three chosen from the terms length, width, height, depth, and breadth, in other words, They are usually labelled *x*, *y*, and *z*. Any three directions can be chosen, provided that they do not all lie in the same plane. In physics and mathematics, a sequence of n numbers can be understood as a location in n-dimensional space. When n = 3, the set of all such locations is called 3-dimensional Euclidean space. It is commonly represented by the symbol $\mathbb{R}^3$. This space is only one example of a great variety of spaces in three dimensions called 3-manifolds. [3.]

2.1.3 Translation, Rotation and Scaling

Any game object whether it is in 2D or 3D can be performed with three transformations: translation, rotation, and scaling. Translation relates to the movements and the position of a game object and is specified by a 2D vector that the person in the previous section moved on the map. The important feature of a translation is whenever the x, y, or z values of an object are modified. The values can be changed all at once with a vector or one at a time. To move a game object in the x direction by 8, the Unity c# is:

```
gameObject.transform.position.x += 8;
```

Listing 1. Object moves in the x-axis

To move the game object by 5 in the x, 9 in the y, and 14 in the z, in the Unity C#, it could be written:

```
gameObject.transform.position.x += 5;
gameObject.transform.position.y += 9;
gameObject.transform.position.z += 14;
or gameObject.transform.position = new Vector3(5,9,14);
```

Listing 2. Translation test file

Unity stores rotations as Quaternions internally. To rotate an object, use Transform.Rotate. Use Transform.eulerAngles for setting the rotation as euler angles. A game object can be rotated about its x, y, or z axis or the world x, y, or z axes. Combined rotations are also supported. An object can be rotated around arbitrary axes defined by vector values.

Finally, scaling changes the size of an object as shown in Figure 4. A game object can be scaled along its x, y, or z axis. It is essential skill to change the scale of the game object by setting each scale value individually, thus:

Figure 4. Scaling an object.

It has to be known that the values for the scale are always multiplied against the primary size of the game object. Therefore, it is illegal to make a scale of zero. If the designers want to flip an object, a negative scaling value could be used. For example, setting the y axis scale to − 1 will turn the game object upside down. It is the necessary skill to understanding how game objects will move around within a game. The designers have to take some time to orient themselves with both 2D and 3D space. Fortunately, Unity provides the feasible mathematics and reveals it behind various easy-to-use functions.

However, when something goes wrong in the game, it should be making some idea where to start watching. [1,198.]

2.2 Design Maxims or Rules of First-Person Shooter Game

The theme throughout the game project is the word mechanic to relate to the actions which are taking place in games from the internal operations of animation and programming to the interactions between the environment and the player. However, in game studies, the game mechanic is used to refer to developed relationships which facilitate and define the game's challenges between game and players. They are complicated systems that contain a series of rational motivations, actions, goals, and feedback of the players. Comprehending a game mechanic is useful for making actions and other elements that may be implemented with that actions open up a plethora of infinite ideas for games by combining actions, rules and goals. The cycle is illustrated in Figure 5.

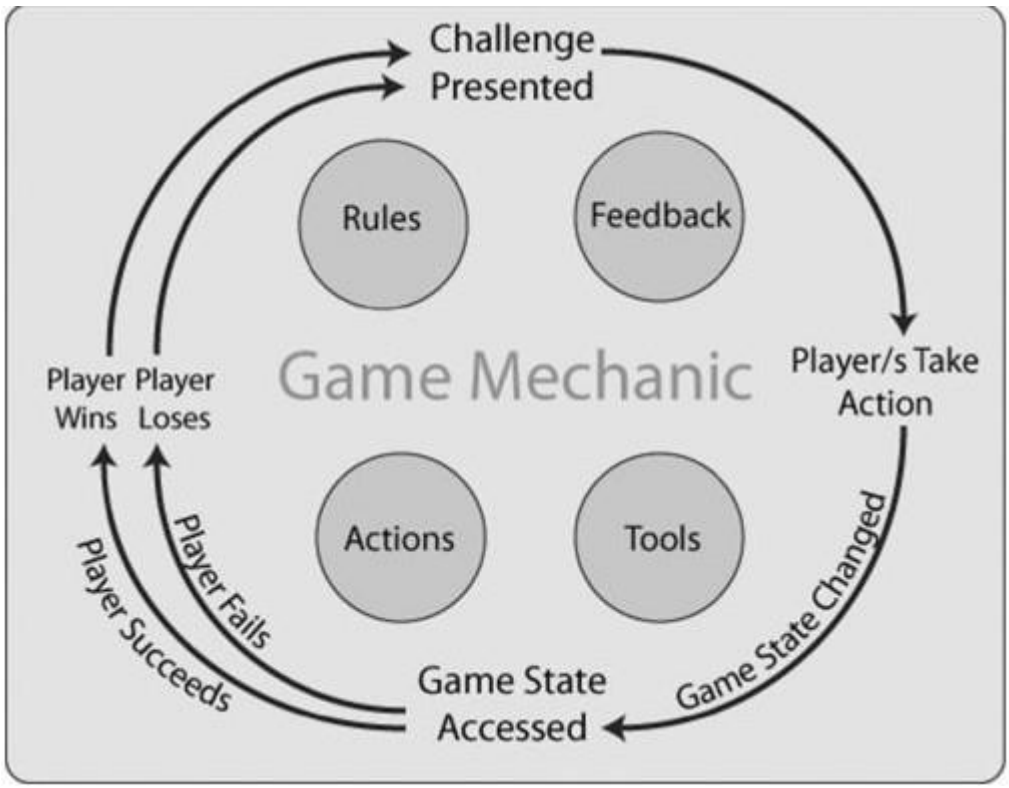


Figure 5. The game mechanic cycle. [1.]

The players are presented with a challenge. To achieve the challenge they obtain tools which can be used to implement actions and rules that can define the scope of these actions. The tools contain peripheral inputting objects such as sensors and keyboards, as well as virtual tools in a game such as weapons, tool-boxes and keys. The rules order how the players can play in the game world. In a card game, generally rules are written in an instruction booklet and handled by players. In a video game, the player is guided to be aware of the rules and the programming code of the game ensures that the player can follow them. Also the program provides information to players depended on their actions to assist them to learn how to play the game better and complete the challenge. Part of the feedback mechanism can also inform the players while they win or fail.

A first person shooter (FPS) is a genre of action video game that is played from the point of view of the protagonist. FPS games typically map the gamer's movements and provide a view of what an actual person would see and do in the game. [14]

In section 2.1, the 2 dimensions and 3 dimensions were explained with coordinates system. This section will deal with the basic game design concepts, specifically those relevant to the demonstration application. Actually, there is no-one to figure out the specific rules for FPS games, but some game designers summarized a few maxims or rules for this kind of game. Every designer who wants to make an FPS game should pay attention to these maxims or rules. [4.]

Rule 1:	Get into the action early

- Attract the player into the world by force; make use of that original confrontation to set the tone. The first impression has to be followed up by developing the tone. Example: Call of Duty 7. At the beginning of the Russian campaign, the speech of the commissars combined with the war-crafts, explosions, and heavy machine guns is passionate, intense, and does not go on forever.

- The player is not allowed to play the game abstractedly, which is a dangerous signal at any point of the game. Example: Half-Life 2. When the introduction appearing the scene of City 15 was much more effectively than the train sequence of Desert 2 from the predecessors of the game, the biased length of

time between take-off board and getting the hook would never have performed in any other game.

This part is the responsibility of the art designer(s). Normally, the art designer(s) makes several pictures and CG(s) to be GUI to attract the player. The exquisite GUI, meaning of CG and fair-sounding BGM are easier to give the player a nice first impression. Actually, this is one important standard to judge if a game is good or not, because the player always believes the feeling which he or she sees or hears.

Rule 2:	The game world is the real world

There should not be only one approach from one position to another one; the player should not feel constrained in their choice. Example: Halo 2. The environments of the open city allows Master Chief to use different ways to achieve his goals, according to rewarding the player for exploring the environment repeatedly rather than doing nothing, while the replay value is adding. High playability of the linear game rapidly becomes predictable and repetitive; making false ways to provide the delusion of free choice just serves to make players angry. [4.]

Rule 3:	The gameplay has to be fun

That the gameplay has to be fun goes without saying. The game elements and mechanics alone do not assure that the gameplay will be fun, so ways of making it fun have to be devised specifically to the targeted user to fulfil rule 3. [5.]

Rule 4:	Let the players realize their mistakes and survive (sometimes)

It is underused that the players realize that they have made a mistake in a moment. Also giving the player more freedom in order to use their judgment for deciding how to play (even if they're wrong) is very important. A punishing system which punishes any mistake would not be done indiscriminately, because it would prevent the player to learn and adapt; they would rather keep on hitting their head against the keyboard. [4.]

The hard levels have to be set in every game. The player must find the skills to solve the problems which he or she encounters in the game. In the FPS game, the player may be slain by enemies over and over until he or she finds out the way to slay or pass over the enemies. Normally, the way is not unique, so survival is quite important.

Rule 5:	Try not to break the immersion

Immersion is the players' emotional involvement, feel of tension and motivational feeling. A simple crash bug or out-of-the-context content could easily break the immersion of the players and easily take players out of immersion. Rule 6 states that the game should be designed to pull the player in and keep them motivated as long as possible with no interruption of the experience. [5.]

Rule 6:	No-one lives forever

When playing a game, the player should feel a sense of urgency and passion; various meaningful rewards should be made for effectiveness and timelines, even if they have taken effect immediately. To achieve the missions just as the rescuing and fleeing, the success rate that hinges upon whether the player acquired enough abilities makes the quality of the rewards that the player would acquire. The rewards could be used to improve the abilities of the player as a hero. Example: F.E.A.R. The Point Man could acquire an amazing ability which can be exchanged in level four to be one room in one second when the NPC that can help him is in danger. [4.]

Rule 7:	The player must always know the objective

In any game, the players should understand what they need to do, even if they do not realize exactly where to go or how to achieve a mission, they will decide upon their action according to the information or clues which they obtained in the previous plot or mission. If a game makes the player keeping on playing, then the game is failed. Making players to wander the buildings for searching the switches, stronghold, or undefeated enemy just serves as a hint that the missions or clues are waiting for the players to achieve. Example: Quake 4. Kane has the shortest time to activate elevators, and find

torso delivery systems. Even there is a fifteen minute detour to be requested for turning on a barrel launching system, which is enough time for his mate to be divorced from battlefield. However, if they neglect their purpose, they can also find the escaping point at the top of ladders or lifts, but the rewards activating torso delivery systems cannot be acquired.

Following the aforementioned set of design maxims or rules helps insure that the intense and simulative game is played in a fun way. However, the game elements and the sense of reality shall further be balanced finely through intense testing to make sure that a FPS game occupies certain market and to be more playability. The strong function of game engine and colourful game contents are playing the pivotal roles in a processing of game design. Thus, a completed game has to be made by a set of the mature designers, which consists of the Lead Designer, Art Designer(s), Programming Designer(s), Product Consultant and etc. Chapter 3 discusses the function of the Unity 3D game engine and Chapter 4 discusses the use of these design rules for a GUI design and game design.

3 Unity 3D Game Engine and Programming Development Environment

Unity 3D is a cross-platform game engine with a built-in IDE developed by Unity Technologies. It is generally used to develop video games for computer platforms such as web and desktop, consoles and mobile devices, and is applied by several million developers in the world. Unity is primarily used to create mobile and web games, but there are various games to be developed for PC. The game engine was programmed in C/C++, and is able to support codes written in C#, JavaScript or Boo. It grew from an OS X supported game development tool in 2005 to the multi-platform game engine that it is today.

Unity is the perfect choice for small studios, indie developers, and those of us who have always wanted to make our own games. Its large user base (over 400,000 as of April 2011) and extremely active user community allows everyone from newbies to seasoned veterans to get answers and share information quickly.

Unity provides an excellent entry point into game development, balancing features and functionality with price point. The free version of Unity allows people to experiment, learn, develop, and sell games before committing any of their hard-earned cash. Unity is very affordable, feature-packed Pro version is royalty free, allowing people to make and sell games with the very low overhead essential to the casual games market.[6, 5.]

3.1 GUI of Unity 3D Game Engine

Unity 3D game engine is one kind of visualized game engines. It integrates Animation Mechanics, Character Mechanics, Player Mechanics, Environment Mechanics and Programming Developer together. Also it supports online assets shop to designers. The designers could find and buy abundant game assets. As well the designers could design their own assets by themselves. In this section, the GUI of Unity 3D is demonstrated.

Main Editor Window

Download Unity from official website and install it. When the user runs it for the first time, he or she will be required to register the product by following the prompts. Registration can be done fast online even if the machine which the user has installed it on is not connected to the Internet. To open with Unity, the user always starts with a project. According to the operating instructions, the Main Editor Window would be opened, which is shown in Figure 6.

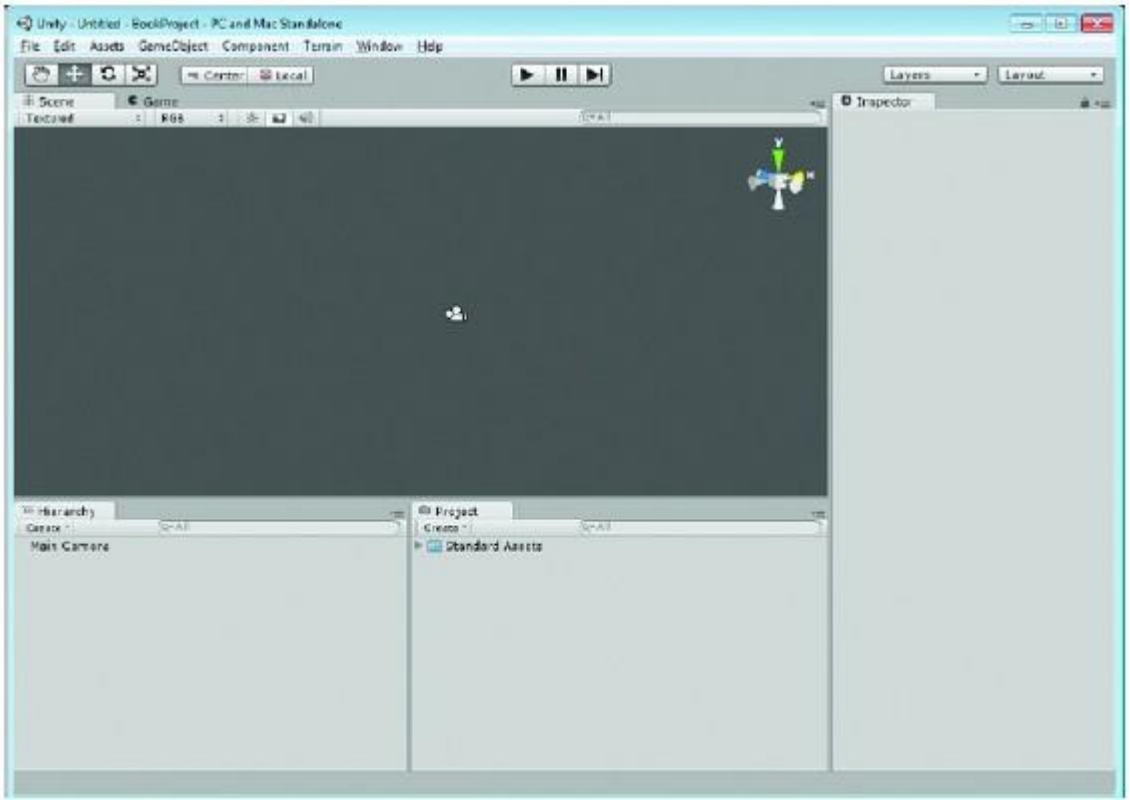


Figure 6. The new project in the Main Editor Window

The Main Editor Window indicates all tools and function windows. The first step of a designer is to know this interface and what every button means, because almost all operations are done in the window except programming edits. Every sub-window should be known well.

Scene Window

The Scene Window mainly stores model assets of the game project. There are various kinds of 3D models, such as players, enemies, NPC, items, sky, and terrain. All of

these models can be displayed in Scene Window. This window is the most important part of Unity 3D, because it is the visualized design window, as shown in Figure 7.

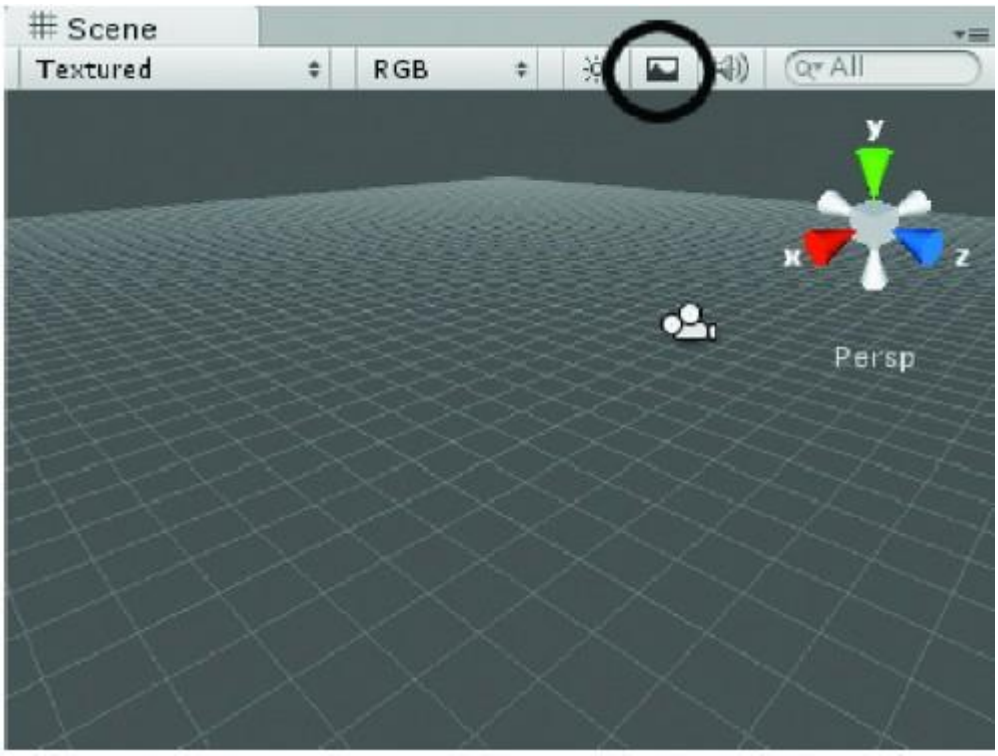


Figure 7. The Scene window indicating the camera object and the grid

The Scene window, shown in Figure 7, is where the user builds the visual scene of the project, which he or she plans and executes their ideas. To show the grid in the Scene window, switch on/off the Game Overlay button which is surrounded with the black circle. The designer could observe the whole structure of the project which he or she created.

Game Window

After every complex edit, the objects reach the space which demonstrates the effect of a game project, which is the Game View. The scene demonstrated depends on the space where the camera illuminates. The designer can modify the scene in the Game view, according to adjusting the position and rotation of the camera.

The Game window, shown in Figure 8, is the visual space where the user tests his or her game object before building it in the runtime environment. It is a simulate runtime

environment, thus all objects have been implemented, unlike the Scene window, it has no default lighting. The designer has to create the Main Camera to see objects in the Game window.

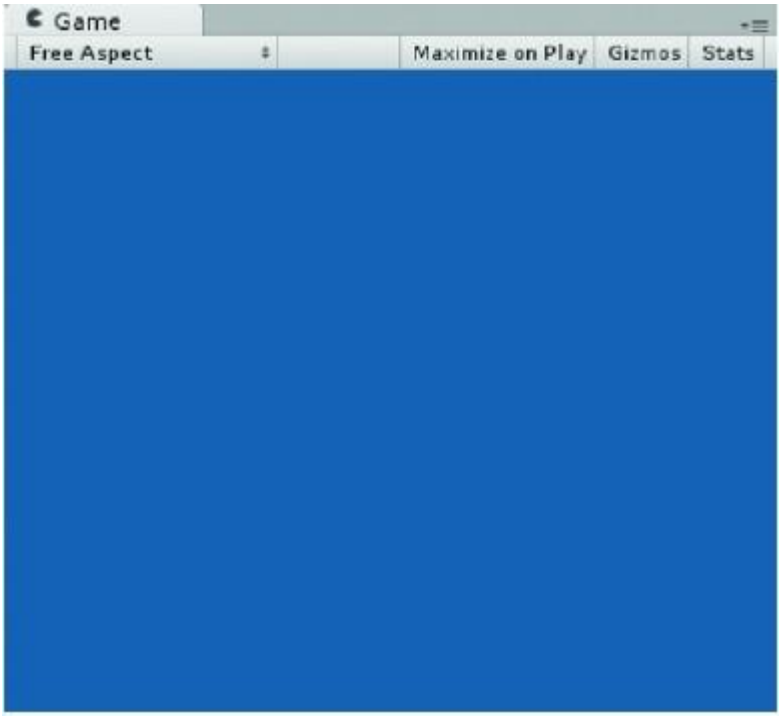


Figure 8. The Game Window

In the Game View, a “Free Aspect” list is used to set the aspect ratio of the game. The “Maximize on Play” button is used to maximize the game scene. The “Stats” button is used to set whether opening the window of the game status or not. The “gizmos” tools bar can indicate all information of tools.

Hierarchy View

The Hierarchy View is used to store the specific objects in the Game Scene, such as camera, texture 2D, texture 3D, light, box, spheres, cubes, models, plan, terrain. After any new game project has been created, it is default to create a game scene and add the main camera to the Hierarchy View of the scene.

The Hierarchy view indicates the game objects in the currently activated scene. Game objects that are dynamically created and removed from the activated scene will emerge here when they are created and activated in the scene.

Project Window

The Project Window mainly stores all asset documents which contain game scripts, prefabs, models, animation models, fonts, physical materials, and GUI skins. The designer can create or delete all material related to the project in the Project Window, which is shown in Figure 9.

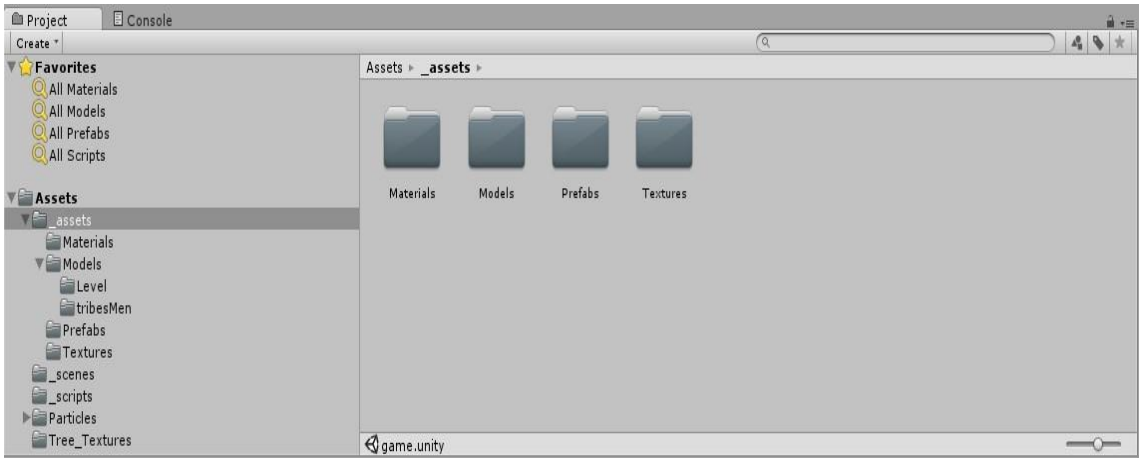


Figure 9. The Project View

The Project view, shown in Figure 9, includes all the assets would be used for the current project, as well as all scenes or levels are available for the completed game or application. It is the storage which stores the assets that could be used for the project. The assets can be deleted from the hard drive according to remove them from this location. However it has to be cautious to remove the assets from the directory in Explorer, because it would cause breakdown of the scene which has activated these assets. [6, 61.]

Inspector

The Inspector view is used to access various properties and components for the game objects which the user has created and selected in either the Hierarchy or Project views. Other object-related information can be accessed here. For example, click the

object that is in the scene view currently, the Main Camera, created in the Hierarchy view, and then watch the Inspector view, which is shown in Figure 10

Figure 10. The Inspector with the camera selected

The Inspector View is more complex relatively. It can be thought of as the space which stores information of the game objects, game assets and game configuration. Whether selecting a game object in the Hierarchy View, or a game asset in the Project View, Inspector View would be opened.

Toolbar

Below in Figure 11 the menu bar is the toolbar, which contains five different controls.

Figure 11. The Toolbar

The Transform tools afford functionality for manipulating and transforming. The button on the far left, the palmate tool, can also become transforming and scale tools for manipulating and adjusting the Scene view. The user can change and move the view of the objects by clicking and dragging them in the Scene view. To revolve around the current viewport object, hold either the Alt key down or the right-key of the mouse while clicking and dragging. To zoom the view, hold the Alt key and the right-key of the mouse, at the same time slide the mouse UDLR (Up, Down, Left, Right). Certainly there are other ways to operate these manipulations, which the user will discover while he or she creates an object into the scene. But do not try it until an object is added to the scene. [6, 63.]

3.2 Significant Objects and Tools in Unity 3D

In section 3.1, the basic UI of the Unity 3D game engine was explained. However most of objects and properties could not be seen before the project was created. A completed game would be made of various objects and scripts. This section would illustrate a few important objects and tools which were used in this project.

3.2.1 Cameras

The camera in Unity is created as a game object in the Hierarchy view and indicated in the Scene window automatically. In a game the camera provides the visible area on the screen, which means the camera provides the height and width of the view, also the depth can be set. The whole visible space by a camera is called the view volume. If an object which is created to the scene is not placed inside the view volume, it cannot be

seen on the screen. The shape of the view volume can be adjusted to orthographic or perspective. These views are constructed from an eye position which could be imagined with the location of the viewer, a near clipping three-dimensional space. [1, 192.]

Unity is illustrated in Figure 12. A perspective camera is used in Figure 12a. The way in which perspective projections best show depth is evident from the line of buildings getting smaller as they disappear into the distance. This is not the case for the orthographic camera shown in Figure 12b. Depth can only be determined by which objects are drawn in front. The buildings appear to be flattened, because there is no the difference of the size between the buildings in the distance. Figures 12c and 12d illustrate the approach in which the view volume of the camera is indicated in the Scene window. If an object is not placed inside the view volume of the camera in the Scene, it will not be seen on the screen in the Game.

Figure 12. A perspective camera in Unity 3D. Copied from Penny de Byl (2013) [1, 194.]

Figure 12 is a 3D game scene in Unity by using a perspective camera (a) and an orthographic camera (b). The tapered clipping shape is indicated by using the perspec-

tive camera (c). The cubic clipping shape is indicated by using the orthographic camera in the Unity Scene.

When a new project is created in Unity, it will accompany with the Main Camera as a game object in the Hierarchy. The settings will be indicated in the Inspector by selecting the Main Camera. While the near and far values which appear in Clipping Planes are set in the Inspector, the resulting frustum can be indicated in the Scene. The width and height of the viewing volume can be changed by modifying the value of the field of view (FOV) for a perspective camera. The larger value of the field of view, the more the player will be able to watch around the area where is immediate. To obtain a feel for the field of view, hold our arms out to the side and look straight ahead as if to make a cross figure with our body. Tardily bring our arms around to our front until we can just see both hands out of the corners of our eyes while looking straight ahead. When the hands come into peripheral vision, the angle which the arms make is the field of view. The average forward-facing field of view of the human is approaching to 180° however various birds are capable of almost 360°. The value of Depth is set for adjusting the complementary field of view for the orthogonal camera in Unity. [1, 195.]

3.2.2 Terrain Editor

One general feature with many game engines is a terrain editor. Not only the users can sculpt their idea into being, but they can develop it with flora from grass to shrub to forest via paint textures. Unity provides a full-fledged terrain editor. In the terrain editor, various objects can be created and developed, such as LOD (level of detail), distance culling, and animations. Although this makes it extremely simple to set up and develop a terrain, as with anything that makes numerous features for them, it also comes with limitations. To acquire more in-depth information on the tree generator, the Reference Manual must be found, and then choose the Terrain Engine Guide.

A Terrain object would be created to the scene in both the Hierarchy and the Project views with the position at 0, 0, 0 when the user presses Create Terrain button. In the Inspector, when the Terrain object is selected in the Hierarchy, the user will see the properties and the tools which are available for developing it, as shown in Figure 13.

Figure 13. The Terrain Tools in the Inspector

A feature must be regarded, which making sure the use using Default rather than Scene is lighting in the Scene window. The position, rotation and scale would be set in Transform section, and here the default value is ok. The tools in Terrain (Script) section are the main tools to create the terrain. For example, the user can use the first button to create mountains with different heights and the fourth button to paint the trees, bush or grassland. In general, the terrain editor is one of strongest functions in Unity 3D. However it will not be illustrated in this paper. The readers could visit some unity forums to find more experiences about the terrain editor.

3.2.3 Skybox

The most general method for creating a sky with cloud textures and weather is to import a skybox. Actually this is an inside-out cube placed over the main camera with seamless images of the sky rendered on it. Because only six planes and six textures are required, it is a relatively cost-effective approach to create a convincing-visual sky.

The six textures are referred to by their position on the cube: up, down, front, back, left, and right. An example skybox is shown in Figure 14.

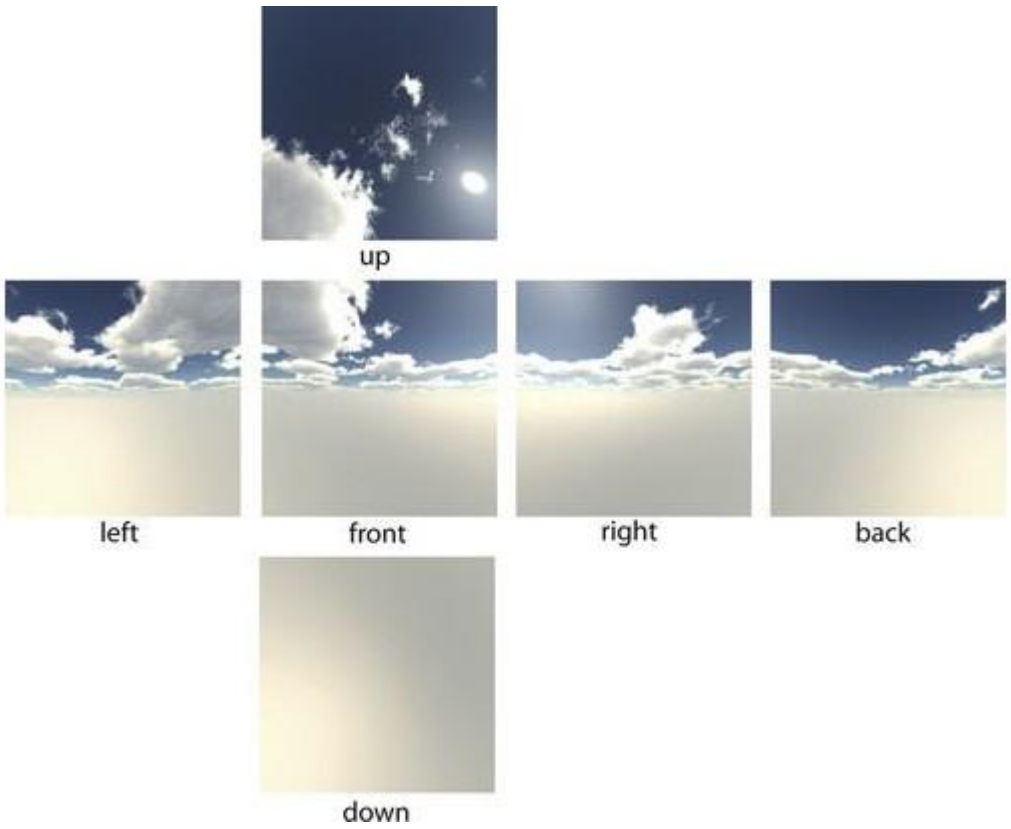


Figure 14. The six textures are used to make up a skybox

Unity provides various skyboxes. To use them, the skybox package has to be imported by right-clicking Asset > Import Package > Skyboxes. To implement a skybox to a game, select the Main Camera object in the Hierarchy and find the location of the Camera component in the Inspector, and then set the value of Clear Flags to Skybox. Finally return to the main menu and select Edit > Render Settings, then the user can choose the skybox material which he or she wants to apply with in the Inspector. In the Game, the default skybox material (none) is applied. [1, 1043.]

Clouds

Dynamic fog is a more complex technique than the prefabricated fog. However the common standard fog applied in computer graphics requires a faded out effect over all 3D assets such as terrain, prefabs in a scene, volumetric fog is included within a 3D space. This requires more intensive processors rather than render. However, this is not

a real issue on current consoles and desktop machines. The types of natural effects that can be completed with volumetric fog contain dark clouds, dust, and mist. The player can watch the clouds whenever he or she wants to watch them from above. Whereas the default fog provides a set of density, in volumetric fog the player can walk through dense patches and light patches.

Weather

The weather impacts the visual sense and feel of the game environment. Although players will not be able to feel the moist of virtual rain in a game environment, the appropriate lighting, colouring, and special effects such as particle collider will make the sensation like they are personally on the scene. [1, 1047.]

In this section, several important objects and tools were explained and illustrated. However, a completed game is made of various objects and assets. Every excellent game is completed by dozens of designers and it is always accompanied with abundant assets and functions. In this paper, a few frequently-used objects and tools were explained, which can be used to make a playable game.

3.3 Development Environment and Scripts

The core of interactivity is scripting. In Unity, every action and functionality is required to be scripted. Fortunately, Unity provides plenty of ready-made scripts from various sources. Unity operates with a large collection of useful scripts to get one up and running. The Help button indicated in the main menu is full of examples and segments of scripts; a number of full Scripting sections also can be found. The Unity Forum and Unity Answers also obtain various useful sources. As well the user can download the tutorials in which he or she wants to find the useful approaches from the Unity website. It would better to use the tutorial resources which contain the excellent cases with explaining what the code does. Unity scripts can be programmed in C#, JavaScript, and Boo. [6, 100.]

Scripts

Scripts consist of four main types of components: variables, functions, equations, and comments. Variables hold values that can be anything from numbers to text. Functions

do something, generally with variables and equations. Comments are ignored when the code is executed, allowing the user to make notes about what the code is or should be doing or even to temporarily disable the code.

When the user creates a new script with C#, a new script file would be created in the area Assets>Script which is a folder that could be named by the user. If the user double-clicks this file, and then the new script would be opened with MonoDevelop as a new C# programming file. The MonoDevelop adds the necessary head files and classes automatically, as shown in figure 15.

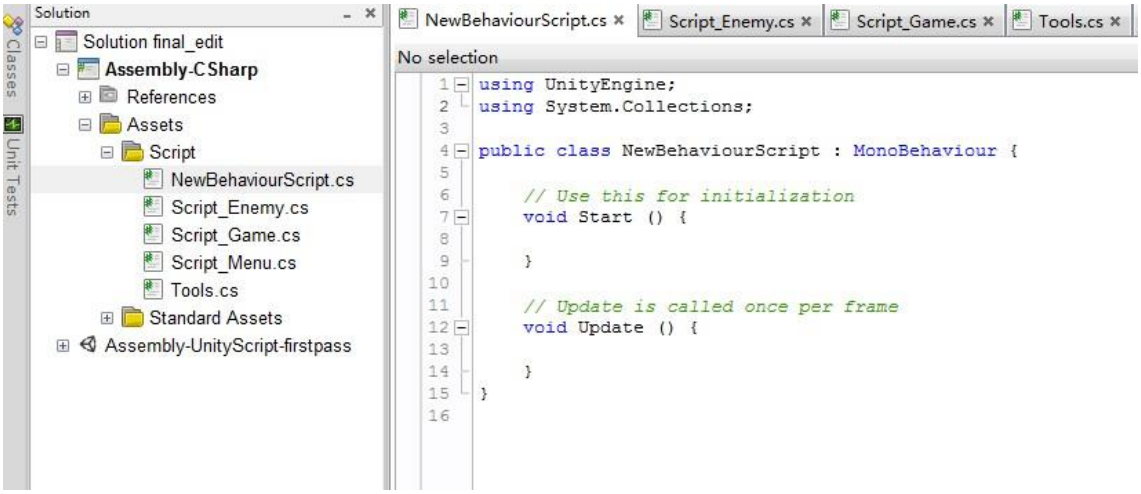


Figure 15. The new script

The function named Update is one of the most important functions in any game engine. This function which is one of the system functions is checked at least every frame during runtime to detect whether anything needs to be executed. For example, it can implement the animations when a particular condition is met by checking.

The scripting system is based on Mono, [7] an open-source version of .NET. Like .NET, mono supports many programming languages, but Unity only supports C#, Boo, and JavaScript. Each language has its pros and cons. C# has the advantage of having an official definition and many instructional books and online tutorials, is the closest features to a main language on .NET and Mono, and thus used in plenty of commercial and open-source software. But C# is the most verbose choice. In this game project, the script language was C#. The game project included four scripts. Figure 16 shows all scripts of the game.

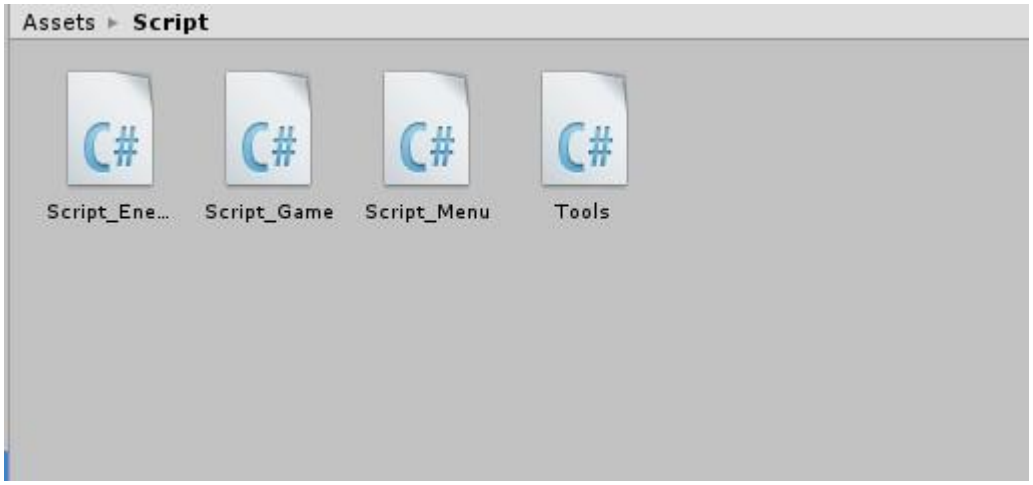


Figure 16. All scripts in this game

All scripts were created with the JavaScript language. These scripts must be bound with special prefabs or game objects. The version of C# in Unity is more succinct but apparently derived from the Boo implementation and not quite standard C# script (so it has been suggested that it be called UnityScript).

In this chapter, the Unity 3D game engine was introduced and the UI was indicated. The Scene view is the environment of making a game, because it is the visual environment, and it supports a huge convenience to designers. Every view and window plays the role of tools which could decorate the game. The parameters of every object have to be set reasonably. The implementation of the game is script. Every effective script must be bound with object or prefab, in order for the object to be working.

4 Implementation

The previous chapters dealt with the basic theoretical background about 3D game, and Unity game engine and its development environment. To demonstrate how a 3D game can be developed, a case study called First-Person Shooter (FPS) game was specifically created. This chapter demonstrates and discusses how the game and its main features were implemented and it deals with design, methods and algorithms.

The player plays a sniper to find out all enemies and slay them on a terrain. When the player shoots to enemies, the enemies generate hatred and shoot the player. Both the player and enemies obtain heal-points (HP), who is slain when its HP is below zero. The player could be healed when he or she touches the heal-boxes. The player must slay all enemies to finish the game.

4.1 Collection and Design of Game Assets

A completed game is normally made with the required game assets. The game assets consist of art assets and script assets. Art assets contain 2D/3D models, textures, pictures, music and etc. They are usually used to make game objects and Graphic User Interfaces (GUI). Game objects are the objects which can be in sight and arranged in the game scene. It can be said that game objects are the essential elements which constitute whole game. And script assets are used to make all game objects to be performed. They are programmed with codes. Every script must be bonded with the suitable object.

In this game, a few textures were collected and drawn to make the Graphic User Interfaces. And the standard assets packages were imported from Unity Assets Package. Actually they are Character Controller Package, Terrain Assets, Skyboxes and Tree Creators.

Character Controller Package

This package contains a 3rd character model which was used to make prefab with enemy game objects, a First-Person Controller which can simulate the animation of player, and few scripts which were used to make model and controller to work. It is men-

tionable that the animations with “idle”, “walk”, “run” and “jump” are achieved for the models.

Terrain Assets

The package contains the tree model and a few textures and rendering materials, such as grass and palm. The designer can make the terrain and render all elements on the terrain with the Terrain Editor which was referred to in the previous section. Certainly, an outstanding art designer can make the assets more wonderful.

Skyboxes

Skyboxes were introduced in the previous section. The main function of skyboxes is filling colourful dyestuff to the sky space in the game. In this case, the basic blue sky and white clouds were filled. Figure 17 shows the effect of Skyboxes in the game.

Figure 17. The effect of Skyboxes in the game

In Figure 17, the blue sky is a background and white clouds are immobile. The effect is the basic effect of default Skyboxes in Unity. The designer can make more effect with 3D animation software and then imports these assets to Unity.

Tree Creator

Tree Creator is just a folder which contains a big-tree model and rendering materials. It is used to make trees on the terrain. These trees can be collided.

A few textures, pictures and music were collected and designed for the game. Most of the textures and pictures were used to make Graphic User Interfaces of the Game Start, and the rest of the assets were used to draw Heal-Point bar and numbers. The assets are shown in Figure 18.

Figure 18. The art assets collected and designed for the game

The art assets are important in a game. These assets determine the qualities of game models and the effects of the vision. How can a game attract the players in vision? It is the goal which all art designers work hard for.

Other important assets in a game are script assets. The scripts are the key that makes whole game to perform perfectly. In the game, the script language is C#. All scripts are listed and shown in Figure 16. “Script_Enemy” controls the animations, artificial intelligence (AI) and damage calculation of enemy objects. “Script_Game” handles the performances of whole game. “Script_Menu” makes the Game Start window in real-time. “Tools” contains few public functions and parameters.

4.2 Construction of Game Level

A game level is a section or part of a game. Most games are so large that they are broken up into levels, so only one portion of the game needs to be loaded at one time. To complete a game level, a gamer usually needs to meet specific goals or perform a

specific task to advance to the next level. In puzzle games, levels may be similar but more difficult as you progress through the game. [8.]

In games with linear progression, levels are areas of a larger world. Games may also feature interconnected levels, representing locations. Each level usually has an associated objective, which may be as simple as walking from point A to point B. When the objective is completed, the player usually moves on to the next level. If the players fail, they must usually try the same level again or perhaps return to the very start of the game. In games with multiple human players, the level may simply end once a limit in points or time has been reached. Not all games order the levels in a linear sequence; some games allow the player to revisit levels or complete them in any order, sometimes with an over world in which the player can transition from one level to another.

A person who creates levels for a game is a level designer or mapper. The latter is most often used when talking about first-person shooters where levels are normally referred to as maps. The computer programs used for creating levels are called level editors. Sometimes a compiler is also required to convert the source file format to the file format used by the game, particularly for first-person shooters. Designing levels is a complex art that requires consideration for visual appearance, game performance, and gameplay. Creation of levels is an integral part of game modding [9.]

In this game, the game level contains one terrain, ten enemy objects and five healboxes. The elements which designing the terrain include:

- Creation and set of terrain: The designer must create ‘terrain’ game object in the Scene Window, and then click ‘setting’ button to set all parameters in ‘Resolution’ area to be suitable values.
- Topography: The ‘Raise/Lower Terrain’ button must be picked to some damage to the terrain, which means the designer can design the topography on the terrain, such as the mountains.
- Paint Textures: It is the painting approach with the topography in a scene. Some textures like the pigment are painted on the terrain until a terrain is achieved such as a mountain. [6, 131.]
- Trees: The Place Trees tool is an important part of the terrain editor is. The designers could create and arrange trees by using the terrain editor. To populate their scene, the Tree Creator must be imported. The effect depends on the effect of the tree model.

- Sky: A traditional sky-dome would have the advantage of working on systems that do not have much shader support, but because it generally uses simple spherical mapping, it's prone to distortion at the top. Take care to use an image that avoids the problem as much as possible. Sky-domes also have the disadvantage of finite geometry. If they are too close, we will see where they intersect the regular scene geometry. If they are too far, they extend the camera clipping plane and increase the possibility of Z order fighting where two surfaces are too close to each other.
- Packages: Packages are collections of game assets which can be imported into a project without significant dependency. In the following section, I will make use of an asset package imported for this project. I may also wish to experiment with an asset package available for download from the Unity3D website, TerrainAssets.unitypackage. This package includes various useful textures, models, and other terrain-related assets. [6, 150]

According to the above processes, the fundamental terrain was completed. It was a rough sketch as shown in Figure 19.

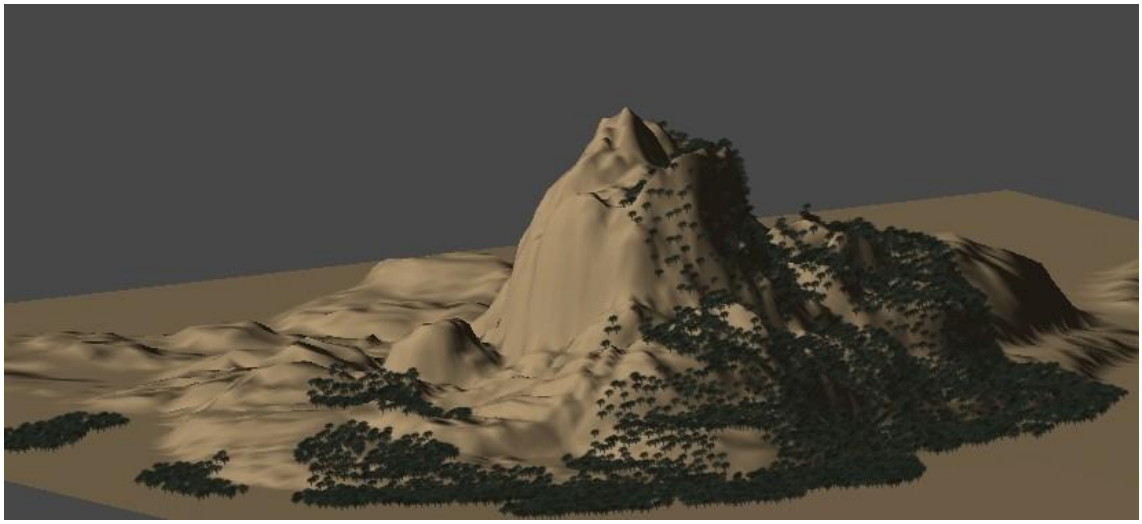


Figure 19. The outline of the terrain

Trees created by the terrain editor may be made to animate to sway in the breeze. The automatic LOD is provided; the full mesh tree is swapped out for a painted board with the image of the tree at a certain distance, and at a far distance, the painted board is not drawn at all, or distance rejected. When the Place Trees tool is selected, the value of these distances can be set in the Inspector, but more importantly, the distances will

be automatically reduced by the engine to insure that the slower machines can also operate with the acceptable frame rate. The shadowing effect is provided. The trunk and leaves will be darkened, when the tree is on the shadowed side of a mountain.

Other game objects containing enemy objects and cube objects were arranged on the terrain also. They constitute a completed game level together. Every game object obtains its own function in the game. Enemy objects must be slain to finish the game, and the cube objects can heal the heal-points of player. In Figure 20, the enemy objects and cube objects are distributed on the terrain.

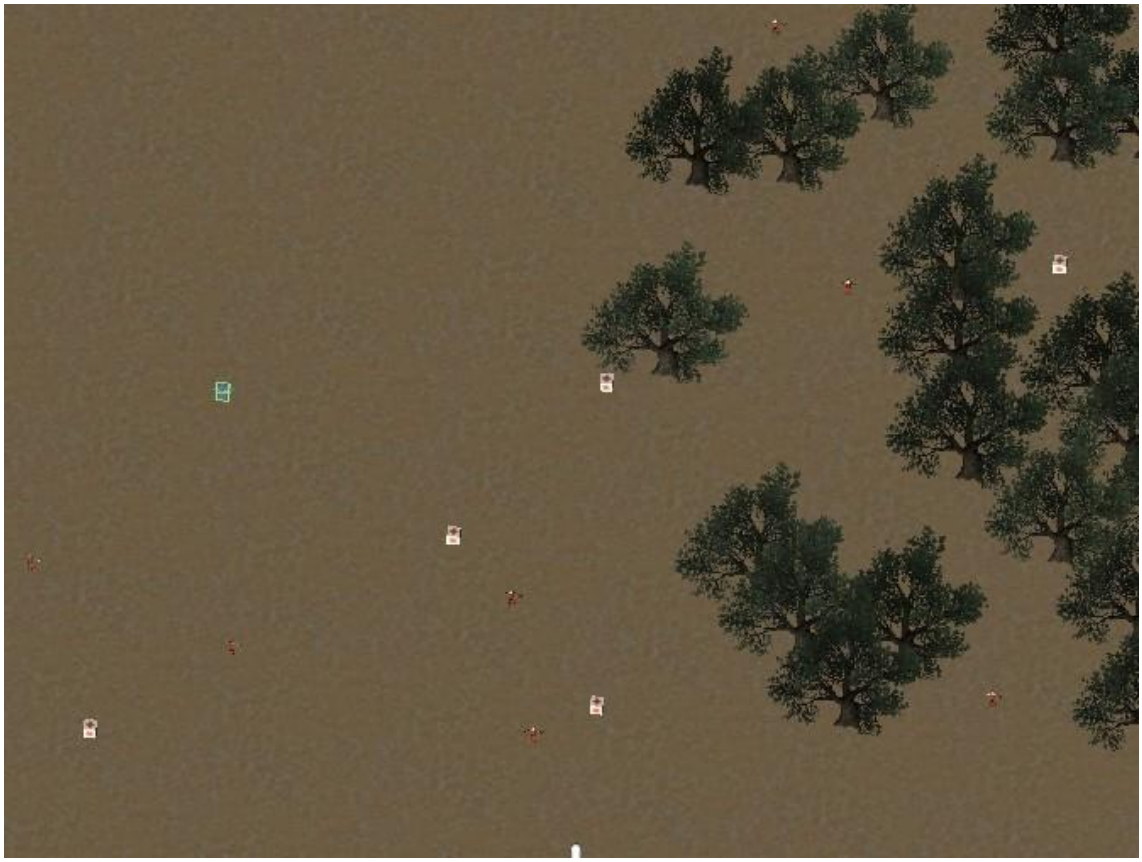


Figure 20. The distribution of enemy objects and cube objects

In Figure 20, a little part of the terrain is shown, and all game objects are distributed in this area. The white points are cube objects and the red points are enemy objects. They distribute around the trees which are green. The scale of the terrain is quite larger than the scales of other game objects, as result all game objects just can be arranged in the little area for shortening the play time.

In the game level, the background music was used. It was easy to be created with a game object and added the audio component with a music source. It should be noted that this game object must bond with the Frist person character controller name “hero” object in the game.

4.3 Implementation of Game Features

Scripts can contain any amount of information and instruction. It will be up to the designer to keep his or her scripts organized, so the game engine will know how and when to use the information. Another scripting challenge is deciding where to put the scripts. With some scripts, for example, the instructions to open and close a door, the choice will be easy; in this case the script can go on any door in the scene that needs to be functional. Likewise, the script that causes an alligator to rush out and snap at the player’s character will more than likely belong on the alligator. Other scripts, however, may not be associated with anything in particular, such as a script that changes day into night. It may control the state of several objects in the scene, but it may be dependent on time, rather than player interaction, to know when to execute the instructions. In this case, the designer would probably make an empty GameObject for the express purpose of holding that particular script and name it accordingly. [6, 93.]

4.3.1 Fundamental Features

In the previous chapter, it was noted that the scripts are the key which makes the game objects to work. The section demonstrates how all the functions of any objects, which were referred to in previous section were implemented with bonded scripts. Four scripts were created with C#. In the Unity game engine, all the game objects that the designer made must be shown in the “Hierarchy” window, and then these objects just can be designed and placed in the “Scene” window. In this section, the fundamental functions are implemented and discussed.

There are two kinds of ways to make game objects:

- Creation: The designers can click ‘Create’ button in ‘Hierarchy’ window to make new and rough game objects, such as ‘Cube’, ‘Sphere’, ‘Camera’ and etc. These game objects must be designed and adjusted by designers to be required objects.

- Prefabs: When the designers find this kind of matter that a same game object has to be used with several times, it is quite troublesome to create the same object repeatedly. Unity game engine supports a way to resolve this condition, and that is ‘Prefabs’. The designers can create game objects with the prefabs in the ‘Project’ window. When the objects are used, the designers just need to drag the prefabs to the ‘Hierarchy’ window.

In the game, two game objects were created with prefabs. They are ‘Enemy’ and ‘Cube’. Next we discuss how to implement the functions of game objects. Firstly, we can see the animations of the ‘Enemy’ object in the ‘Inspector’ window, as shown in Figure 21.

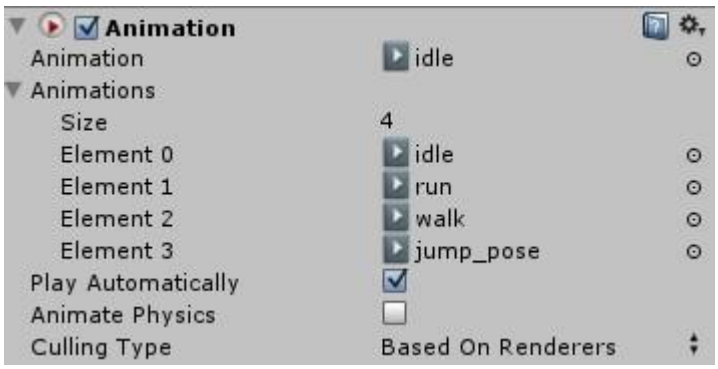


Figure 21. The animations of ‘Enemy’ game object

Figure 21 shows that there are four kinds of animations of this object. They are idle, run, walk and jump_pose. In this game, the first three animations were implemented. It is an easy code to implement these animations with C#. Listing 3 shows the code.

```
if(rand == 0)
{
    //stand state
    gameObject.animation.Play("idle");
    enemyState = STATE_STAND;
}
else if(rand == 1)
{
    //walk state
    //enemies rotate in random
    Quaternion rotate = Quaternion.Euler(0,Random.Range(1,5) *90,0);
    //finish rotate in 1 s
    transform.rotation = Quaternion.Slerp(transform.rotation,rotate, Time.deltaTime*1000);
    //play walk
    gameObject.animation.Play("walk");
    enemyState = STATE_WALK;
}
```

Listing 3. The code to play animation with C# in “Script_Enemy”

In Listing 3, only one function was used to play the animation ‘idle’ or ‘walk’. This code is normally used in conditional clauses. Because the completed codes are long, here the instructions of these animations are introduced.

- Idle: This script makes random numbers from 0 to 2. If the number is 0, then the ‘idle’ is played, and then assigns “STATE_STAND” to ‘enemyState’ parameter.
- Walk: If the number is 1, then the ‘Enemy’ object rotates a random direction in one second, and next plays ‘walk’, and then assigns “STATE_WALK” to ‘enemyState’ parameter.
- Run: There are two ways to make the object to run. One is the hatred is activated and another is attack range is activated. If the player shoots to enemy, then the hatred is activated. If the player accesses to the attack range of enemy, then the result is that the enemy run to player.

The key to engaging interaction is being able to specify conditions under which various alternative scenarios play out. Even in a traditional shooter game, an object may need to be hit a certain number of times before it switches to its destroyed state. Each time it is hit, the program must check to see if certain conditions have been met before it knows what to trigger. Each of these conditions can also be thought of as a state.

.

The game is a Sniper Rifle Mode game which one kind of First-Person Shooter game. Thus there are few required features must be implemented. The Aim Point and the Heal-Points (HP) must be drawn in real-time as shown in Listing 4.

```
void RenderGame()
{
    if(tex_fire)
    {
        //DRAW mouse aim point
        float x = Input.mousePosition.x - (tex_fire.width>>1);
        float y = Input.mousePosition.y + (tex_fire.height>>1);
        GUI.DrawTexture(new Rect(x,Screen.height - y,tex_fire.width,tex_fire.height),tex_fire);
    }

        //draw hero's HP bar
        int blood_width = tex_red.width * HP/100;
        GUI.DrawTexture(new Rect(5,Screen.height - 50,tex_black.width,tex_black.height),tex_black);
        GUI.DrawTexture(new Rect(5,Screen.height - 50,blood_width,tex_red.height),tex_red);

        //draw HP texture
        GUI.DrawTexture(new Rect(5,Screen.height - 80,tex_hp.width,tex_hp.height),tex_hp);
        //draw HP
        Tools.DrawImageNumber(250,Screen.height - 80,HP,texmube);
}
```

Listing 4. The code of drawing AP and HP in “Script_Game”

This sub function is used to draw the Aim Point and Heal-Points bar in the game. The position of mouse must be read and the Aim Point was drawn that depends on it. The textures which were prepared for them were used. The effect is shown in Figure 22.

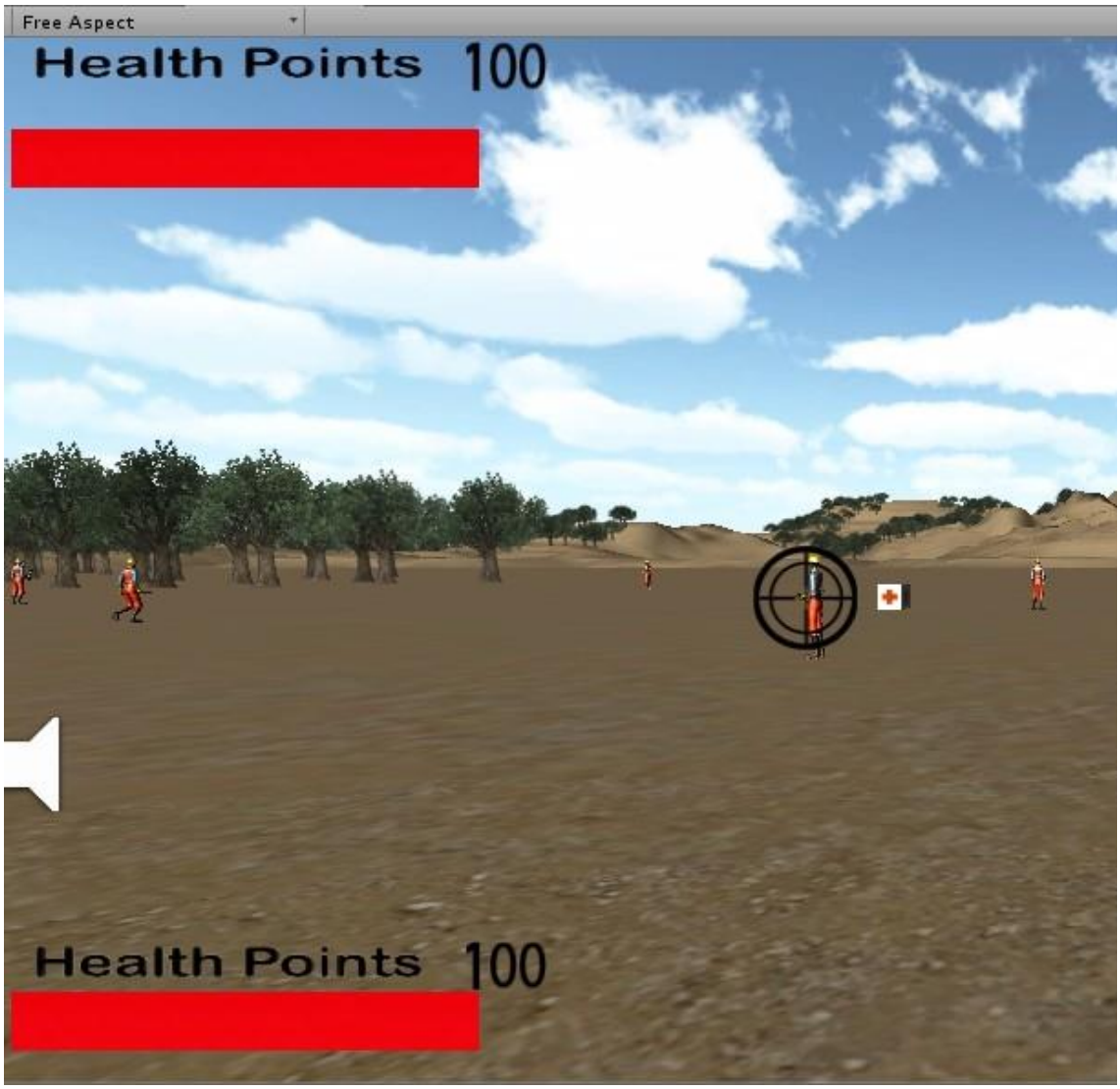


Figure 22. The effect of Aim Point and HP bar in the game

There are two Heal-Points bars to be indicated in Figure 22. The top-left HP bar is HP of enemies. When the player aims at the enemy, it would appear on top-left. The bottom-left HP bar is the HP of the player. It is always indicated until the game has ended.

Another important feature in the game is 'Heal-boxes'. The heal-boxes are the game objects which were distributed on the terrain. Their function is to increase the heal-points of the player when the player finds and touches them. To implement the function, three steps must be considered. Firstly the player must collide with the heal-boxes and secondly the heal-boxes increase the heal-points of the player. Finally the heal-boxes must be destroyed. The relevant codes are shown in Listing 5.

```
void HeroAddBlood()
{
    HP+=50;
    if(HP>=100)
    {
        HP = 100;
    }
}

void OnControllerColliderHit(ControllerColliderHit hit)
{
    //Obtain FPC colliding objects
    GameObject collObj = hit.gameObject;
    //if it is box
    if(collObj.name == "Cube")
    {
        //hero is healed
        HeroAddBlood();
        Destroy(collObj);
    }

}
```

Listing 5. The code of implementation of Heal-boxes in “Script_Game”

The name of the game object is “Cube”. It must be according to the value of “collObj.name”. Otherwise there will be the errors. The heal-boxes are destroyed after they heals the HP of the player one time, which means every heal-box only can be used one time. Thus the HP of the player is limited. This way is one way to control the difficulty of the game.

In Listing 4, it was referred to how to make the graphic user interfaces with C#. The function ‘GUI.DrawTexture(new Rect position, texture image)’ was used to draw texture in game. The function is also used to make “You Win” and “You Dead” GUI when the player wins or dead respectively.

This section demonstrated how to create the game objects, and how to implement the animations of the ‘Enemy’ game object, draw the Aim Point and Heal-Points, and make the Heal-boxes. The fundamental features of the First-Person Shooter game were implemented. The next section will demonstrate and discuss the damage system. It is a special design of the damage for the sniper rifle mode game.

4.3.2 Damage System

Hit points, also known as health points (or HP), damage points, heart points, life points, or just health (among other synonyms), is a finite value used to determine how much damage (usually in terms of physical injury) a character can withstand. When a character is attacked, or is hurt from a hazard or fall, the total damage dealt (which is also represented by a point value) is subtracted from their current HP. Once their HP reaches 0, the character will be unable to fight. In role-playing games, health is often abbreviated as "HP".

The damage system is used to control the HP of the player and enemies. There are three normal ways to design the damage system in a game. [10.]

- The classic damage way: A fixed amount of HP and if they are drained, the role of the player is dead, this is the most simple one and not very realistic, since the role is the same with 1 HP as the role is with 100 HP, but if the role has only 1 HP left every little thing will kill the role. Picking up health packs will instantly heal you. All in all not very realistic.
- The more arcade like one: In some modern shooters the player has regeneration and hidden HP, so if the player gets hit the screen becomes red, but if he or she waits a little time the role of the player will recover and so only die, if the player receive serious damage in a short time, but in the long run the player can eat as many bullets as he or she wants also not realistic.
- Realistic damage system: Direct hits mostly cause instant death and grazing shots will seriously injure the role making the player loses health over time or render the role immobile. Very realistic but also not fun in many cases.

In this game, the classic damage way supported the damage system. There are two different damage ways. One is Player Hurts to Enemies, and another one is Enemies Hurt to Player. There are a few codes to be shown for explaining these two ways. The Player Hurts to Enemies is shown in Listing 6.

```
void OnMouseDown()
{
    //hit enemy
    if(HP > 0)
    {
        //reduce HP
        HP-=5;
        //hit feedback
        transform.Translate(Vector3.back * 0.05F);
        //attack state
        ishatred = true;
    }else
    {
        //enemy die
        Destroy (gameObject);
        //make message of die
        hero.SendMessage("EnemyHurt");
    }
}

void OnMouseUp()
{
    //restore hit feedback
    transform.Translate(Vector3.forward * 0.5F);
}

void OnMouseOver()
{
    //start to draw HP bar
    showBlood = true;
}

void OnMouseExit()
{
    //end to draw HP bar
    showBlood = false;
}
```

Listing 6. The damage way of the Player Hurts to Enemies in “Script_Enemy”

In the previous section the Aim Point was drawn in the game. The position of the Aim Point is accordant with the mouse. Thus if the player aims at the enemy and left-clicks mouse, the hatred of enemy is activated and the HP is reduced by 5. When the HP of enemy is equal or less than 0, then the enemy is slain. The ‘Enemy’ game object must be destroyed. And the script makes a message called “EnemyHurt”. It is the key to calculate the number of the enemies in another script. At the same time, the HP bar of enemy is drawn in real-time because it got damaged. The cycle of drawn HP bar is a completed left-click of mouse finished. The result is shown in Figure 23.

Figure 23. The effect of enemies' HP bar when it got damage

This damage way is a normal way which is used in every game that obtains a damage system. The Enemies Hurt to Player damage way is a different way, because the player must be hurt after the hatred of the enemy is activated. Listing 7 shows how the hatred of the enemy is activated.

```
if(Vector3.Distance(transform.position,hero.transform.position) < AI_ATTACK_DISTANCE || ishatred)
{
      //Enemy goes to run state
      gameObject.animation.Play("run");
      enemyState = STATE_RUN;
      //face to hero
      transform.LookAt(hero.transform);
      //Attack hero in random
      int rand = Random.Range(0,20);
      if(rand == 0)
      {
         hero.SendMessage("HeroHurt");
      }
```

Listing 7. The code used to activate the hatred in "Script_Enemy"

If the player enters into the patrol range of the enemy or shoots the enemy, the hatred of the enemy is activated. The script makes random numbers from 0 to 20. When the number is 0, then it makes a message called "HeroHurt", which means the player is hit. This message is grabbed by a function in "Script_Game", and then the damage is calculated. This way which used the random numbers is one way of the critical damage. In other words, the enemy makes damage to the player with a 5% rate. This way can be used to control the damage speed of the enemy. Listing 8 shows how to finish the damage.

```
//hero is attacked
void HeroHurt()
{
    HP--;
    if(HP <=0)
    {
        HP = 0;
        gameState = STATE_LOSE;
    }
}
```

Listing 8. The code calculating the damage in “Script_Game”

The damage which the player got from the enemy is 1 HP every hit. If the HP of the player is equal or less than 0, then the value “STATE_LOSE” is assigned to the parameter ‘gameState’, and finally the texture “YOU DEAD” is drawn.

When all scripts are completed, these scripts must be bonded with the objectives which are the game objects or prefabs. Firstly the aim game object or prefab must be picked. According to clicking ‘Add Component’ button in the “Inspector” window, the designer can add abundant assets for this object or prefab, such as Scripts, Mesh, Audio, Rendering, Physics and etc. And then the debugging must be done. Unity supports a play mode in real-time, in order to debugging the scripts in any time. The designer can click ‘play’ button above “Scene” window to check whether the script works. If there are any errors in the scripts, then a warning word would appear in the “Scene” window. The designer can pick “Console” window which is right to “Project” window to check what kinds of errors existed in the script. The game objects will be worked, when the scripts are bonded with them.

This section discussed how the damage system was constructed for the Sniper Rifle Mode shooting game. There were two different ways to calculate the damages. A special random damage was implemented when the enemies made the damage to the player. We can understand it does not hit the player with every shoot. This way can be used to control the reduced speed of the HP of the player effectively. Finally, the approach how to bond the game objects with the scripts was explained, and the debugging way was explained also. The game part of the game was demonstrated completely. The game start part will be shown in the next section.

4.4 GUI of Game Start

GUI is a very important component of a game. Unity provides two functionalites called GUIText and GUITexture for 2D text and textures respectively, which are used to display the texts or textures on the screen when the run button is clicked. For example, Fugu Maze on the iPhone has a GUITexture representing a move-forward button on the lower-left portion of the screen. [11, 130.]

To design UnityGUI text, the concept of cascading style sheets (CSS) must be understanded, because it is patterned. The designer can define colours and fonts for a particular style by setting GUISkin. In GUISkin property, every GUI element must inherit its properties. This makes for a consistent look and feel throughout. Also the GUI elements refer to 3D objects can be treated differently.

The default font in the operating system is used firstly in Unity. Fortunately the designer can use any other font which he or she provides. It makes the file size smaller by using the default font. However there would be a small risk that the text could appear variation on different systems. For this game, the font Arial is set which is shown in Figure 24.

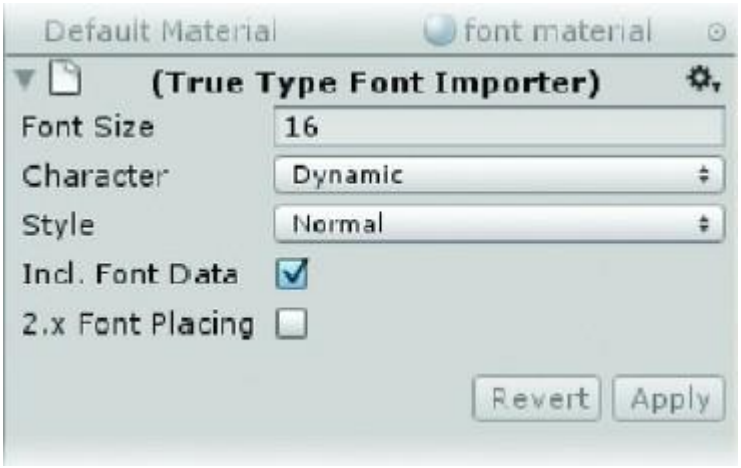


Figure 24. The Arial font in the Inspector

When using a font in most 3D engines for 2D text, a bitmap of the full alphabet and characters is generated for each font and size specified. When the designers specify Dynamic in Unity, the bitmap generated by the operating system is used, saving download time and texture memory during runtime. The downside is that if the designers

specify a font that is not present on the operating system, a fallback font will be used in its place. Even when using generic fonts such as Arial, there may be slight differences among systems. I will investigate this further when we experiment with different fonts later on in the project. Dynamic fonts are currently supported only on desktop platforms (i.e., Mac and Windows). [6, 459.]

In the game, the 2D GUI was made. The assets contain one background picture, one background picture with help information, a few textures with guide text and background music. A new scene must be created firstly. And then the Main Camera object and Music object were created. The position of Main Camera did not need to adjust, because the imagery just is required. In the script, few variables must be defined and initialized to be called in late. These values of these variables were normally initialized with textures in “Inspector” window. The initialized variables are shown in Figure 25.

Figure 25. The initialized variables in Script_Menu

There were three background pictures to be used in the GUI, and an audio source was added to the GUI. The variable which initialized with GUISkin must be defined with “public GUISkin” in the script. In Unity, there are two types of GUI elements. Of the objects that can be created through the GameObject menu, the designers have already used the GUI Texture. The GameObject varieties of text and texture objects are handled individually and exist in the game hierarchy. They are not affected by the GUI Skin specifications. The second type of GUI elements, Unity GUI, must be scripted to exist. If the designers are a bit more comfortable with scripting, they will be using the Unity GUI to create and handle text and to eventually convert the cursor. The Unity GUI consists of a GUI Skin and GUI Styles.

Firstly, a new GUISkin asset must be created according to clicking the “Create” button in the “Project” window, and then pick it, the information is shown in the “Inspector” window. The “Custom Styles” option has to be found, which is very important to help the designers to set their own GUI. The designer must know how many GUI buttons he

or she wants to make clearly. Finally the designer has to enter the number for the size parameter and set his or her own GUI buttons. Figure 26 shows the result.

Figure 26. The Custom Styles of GUISkin in the game

There are seven GUI buttons in the game. Every button obtains its own background picture which was designed for it specially. The names must be made reasonably, because they will be called in the script. It is should note that before the designers experiment with different textures in the GUISkin, they should know that if the texture does not show up in the asset browser, they would not be able to get it back without creating a new GUISkin.

For showing GUI on the screen, the function OnGUI must be used in the script. The `OnGUI` function is similar to the `Update` function in that it is called at least every frame and that is where the code is must be to create the GUI controls. There were four functional buttons required to be implemented. They were Start, Option, Help and Exit. The Start button can start the game when the player clicks it. The Option button obtains two functions: music on and music off. The player can choose if the background music is on or off. The Help button indicates the information of operating ways and the author. The Exit button is used to exit the game. Thus the four functions must be made in the script, and they must be called in the OnGUI function. Listing 9 shows how the functions were called.

```
void OnGUI()
{

    switch(gameState)
    {
        case STATE_MAINMENU:
            //draw menu GUI
            RenderMainMenu();
        break;
        case STATE_STARTGAME:
           //draw start
            RenderStart();
        break;
        case STATE_OPTION:
            //draw optin
            RenderOption();
        break;
        case STATE_HELP:
            //draw help
            RenderHelp();
        break;
        case STATE_EXIT:
            //draw exit
        break;
    }


}
```

Listing 9. The code of the OnGUI function in Script_Menu

The player can activate the function which he or she wants to click. We can see the function “RenderMainMenu” in Listing 9. This function is made to rendering the main menu of the game start. The whole outline of the GUI is designed and made in this function. If the designers are not planning on limiting the screen resolution, the user will be able to choose a resolution at run-time. If they remember that GUI elements retain their pixel size regardless of screen resolution, it becomes clear that they need access to the screen size in order to centre the text messages. Fortunately, they can use Screen.width, Screen.height, and a bit of math inside the command to get what they want. Listing 10 shows the code how the main menu is rendered.

```
void RenderMainMenu()
{
    //set GUI skin
    GUI.skin = mySkin;
    //draw BG texture
    GUI.DrawTexture(new Rect(0,0,Screen.width,Screen.height),textureBG);
    //START button
    if(GUI.Button(new Rect (0,30,623,153),"","start"))
    {
        gameState = STATE_STARTGAME;
        Application.LoadLevel ("Scene_Game");
    }
    //OPTION button
    if(GUI.Button(new Rect (0,180,623,153),"","option"))
    {
        gameState = STATE_OPTION;
    }
    //HELP button
    if(GUI.Button(new Rect (0,320,623,153),"","help"))
    {
        //HELP state
        gameState = STATE_HELP;
    }
    //EXIT button
    if(GUI.Button(new Rect (0,470,623,153),"","exit"))
    {
        Application.Quit();
    }
}
```

Listing 10. The code of RenderMainMenu function in Script_Menu

In the previous section, the function “new Rect” was used to set the position, height and width of the texture. The function “GUI.Button” is similar to the function “GUI.DrawTexture”. However the last parameter of the function “GUI.DrawTexture” is using the texture immediately, and here the Custom Styles which were set in “GUISkin” are using in the “GUI.Button” function. Another special function is Application.loadLevel. This function is used to load another game scene. It is the secret that when the player clicks the Start button, then the game is running.

In the previous paragraphs, almost all functions which were used in the script were listed and explained. The key points are the calls of the textures and outline of the GUI. The next the effect is shown in Figure 27.

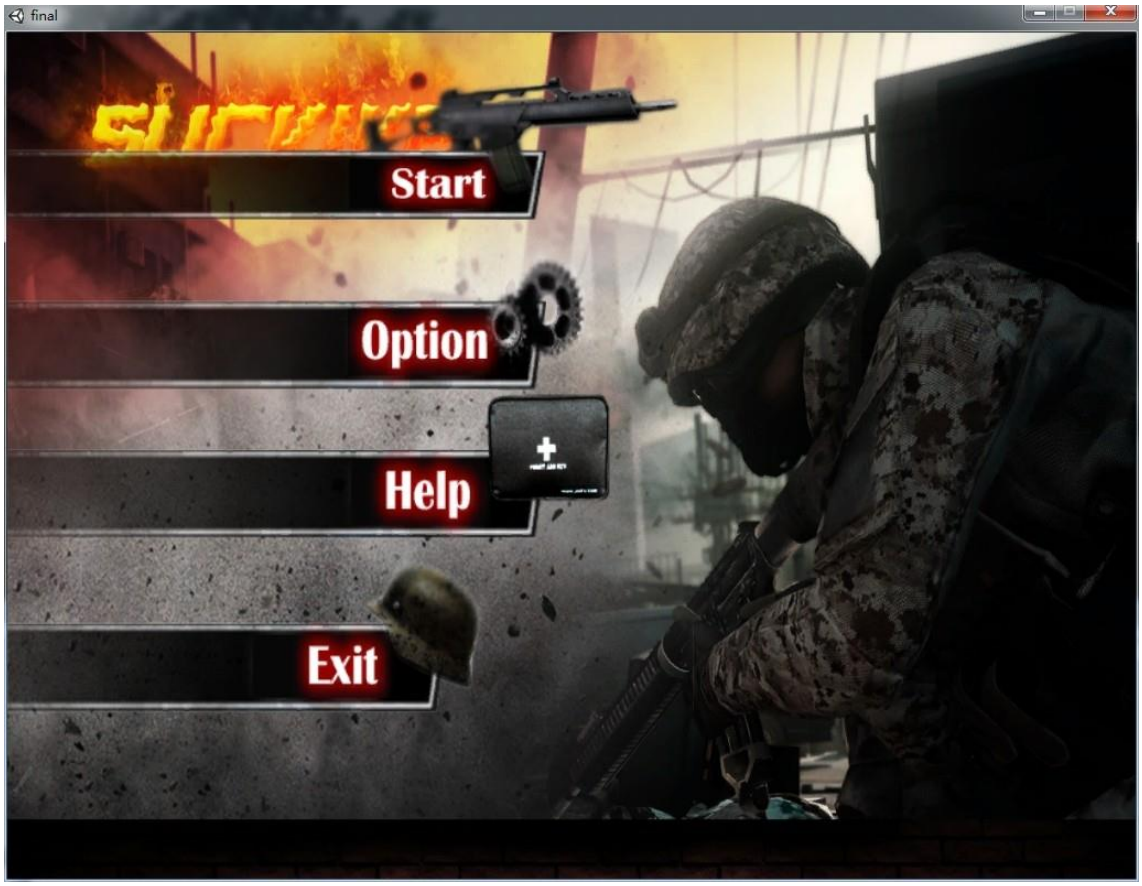


Figure 27. The GUI of the game start in the game

The process which is to design the GUI requires more patience. The textures and pictures require collecting and designing to be necessary. However it offers the sense of achievement to the designers.

In this section, the Unity GUI was introduced as a means of adding 2D text on screen to display mouseover and message text. I discovered the GUISkin, which provides a way of creating GUI elements with a consistent look and feel, and later, the GUIStyle, which is a means of overriding the skin in particular places. I experimented with the parameters for the GUI controls, and found that they could make label controls look like box controls while still acting like labels. After I learned that the GUI elements exist only through scripting, I found that “Rect” parameters define the position and region a control will occupy. I would find more fun designing a game or a brilliant GUI.

4.5 Building and Running the Game

The Unity game engine is a cross-platform engine. It is limited with the interfaces among the different game platforms. Unity game engine supports almost all game platforms in the world, but because of the file size of the executed files, the games designed with Unity are released on computer platforms. The game is also built on the Windows 7 system. It is quite easy to build a game in Unity. The designers just need to click the “File” button and pick the “Build & Settings” button to finish it. The interface of the “Build & Settings” is shown in Figure 28.

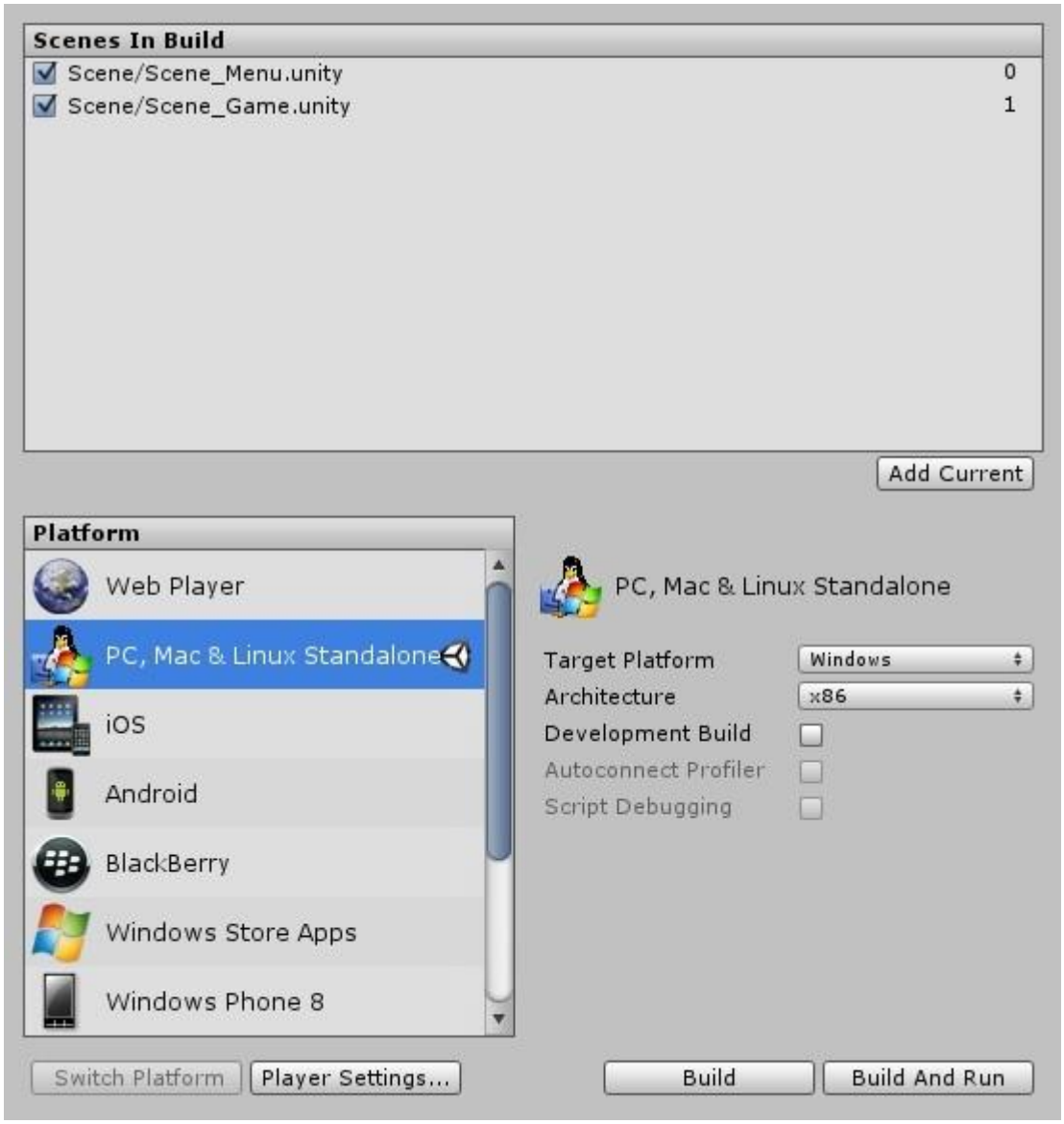


Figure 28. The interface of Build & Setting in Unity

In Figure 28, the Scenes in the “Build” window are shown. I must add all scenes which I have designed for the game. And the scenes have to be marked. Then I can choose the platform which I want to release the game on in the Platform window. On the right, the information of the platform which I chose is indicated. After everything is ready, if I click Build or Build and Run button, the game will be made to the exe file. The players can double-click this exe file to play the game. The size of the game is 20 MB. It is a little game, but it is fun.

In this chapter, the whole process how the game was designed and implemented was demonstrated. The damage system of a FPS game was discussed also. The parts of the codes were demonstrated and explained. The several functions are the fundamental functions for designing a game, such as OnGUI, GUI.DrawTexture, gameObject.animation.Play and etc. Unity supports the strong functions to the designers.

5 Usability Test

Software projects have four objectives; to produce the required functionality, in budget and in schedule, with acceptable quality. That statement may be true for ordinary software development projects, but are these objectives enough for game development. [13]

Playability is the ease by which the game can be played or the quantity or duration that a game can be played and is a common measure of the quality of gameplay. Playability evaluative methods target games to improve design while player experience evaluative methods target players to improve gaming. There are various approaches to test how the players would feel with a game. In the final step of the project, the survey was made to test the playability. There were ten players to be surveyed. The basic information of them is shown in table 1.

	Degree Programme	Degree	Gender
Description	Information Technology * 7	Bachelor * 8	Male * 7
	Business Management * 2	Master*1	Female * 3
	Neurology * 1	Doctor * 1	

Table 1. The partition of the players (* means the amount of the people)

I made the survey with the ten players after they had played the game. The goals of the survey are to test the functionality and playability. The survey contains three stages.

Data Collection

A questionnaire was created to ask few questions about the completeness of the functionality and playability (see Appendix). The first part was used to collect the information of the players. The second part had shown the functionalities that we discussed in previous chapters. These functionalities were listed on the questionnaire paper, and the scores between 0 and 2 were appended. Score 0, 1, 2 was respectively represented uncompleted, completed and completed well. The list includes the First-Person

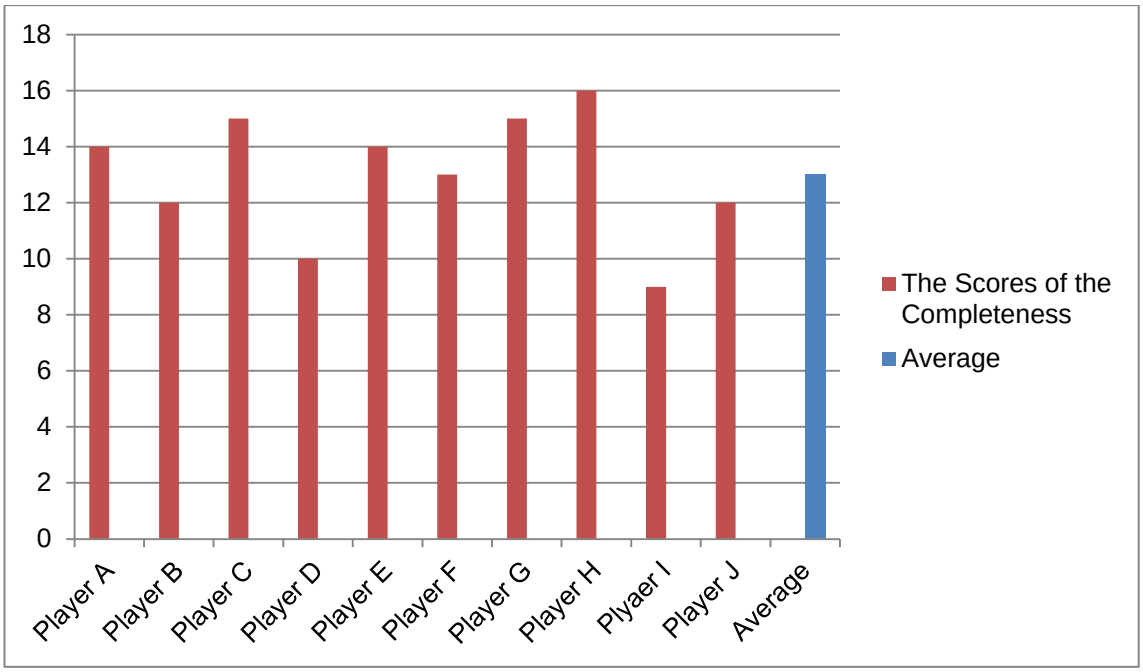
Controller, the Shooting Mode, the Terrain, the Skybox, the AI of the Enemies, the Damage System, the Heal-boxes and the GUI, which represents eight functionalities in the game. The every player could make score with the functionalities up to 16 points. The third part was the question about the playability of the game. The question supported three options to the players. The first option was the game is quite easy to play, it is a recreational game and not worth selling or leaning. The second option was the game obtains some useful functionalities, it is enough used for education. The last option was the game is worth to be the product of business. Finally, the players could make suggestion with the game.

Date Analysis

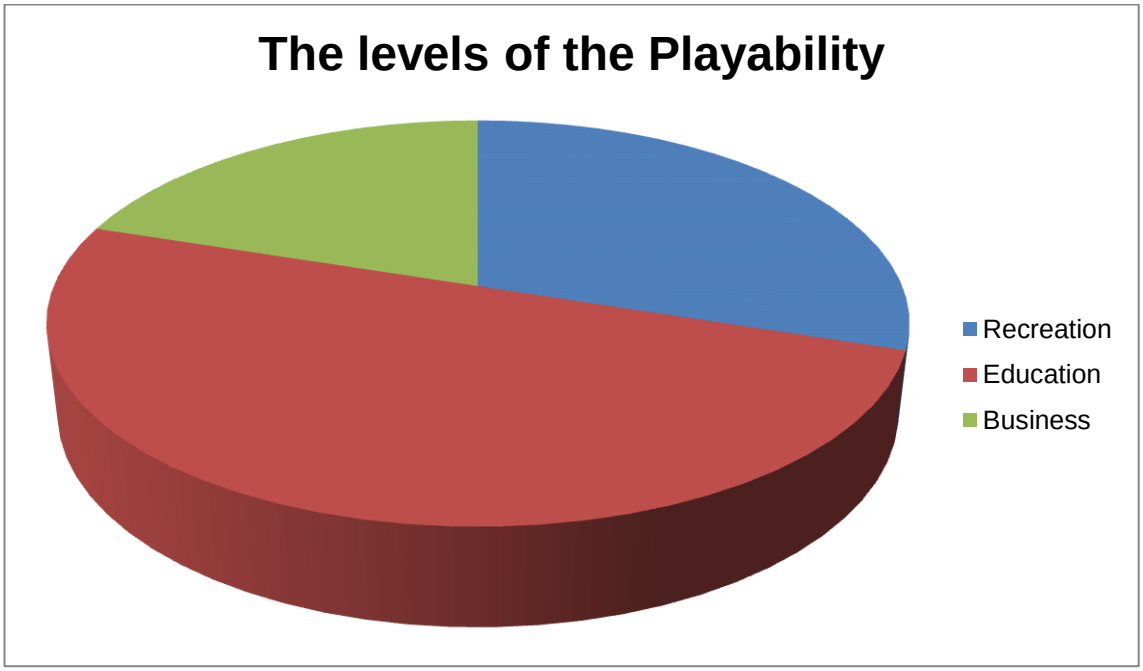
According to the survey I have got the amount scores of the completeness of the game and the level of the playability. The scores were divided to three levels, 0 – 7 was bad level which means the game was not completed, 8-12 was good level which means the game was completed, 13-16 was the favourable level which means the completeness is high. The proportion that every option was chosen in the part Ⅲ indicated the playability of the game. I think any option that was chosen by 40 percent players could represent the result.

Result

The result of the survey is that every player made more than 8 points in the part Ⅱ, and 50 percent players chose second option, 20 percent players chose the last option and 30 percent players chose first option. The both results are shown as in table 2.



(a)



(b)

Table 2. The Results of the survey

According to the result of table 2 (a), we can find that the average score is 13, which means the players who participated in the survey considered the game has been achieved the premier goals and completed well. The table 2(b) indicated the level of the playability, most of the players thought the game could be used for education.

6 Results and Discussion

6.1 Drawbacks and Future Development

Currently the game is achieved completely as a First-Person Shooter game on the Windows 7 platform. The player can enjoy the fun when he or she shoots the enemies. Because of the damage system, the player would be killed by the enemies. He or she would play the game seriously.

The disadvantages are obvious also. The game only support one shooting mode which is the sniper rifle mode, which means the player cannot choose the weapon that he or she favours to play the game with. The new play mode means more weapon models and damage systems required. The design of 3D model requires the professional talent who is good at art design, or the models can be found and downloaded from the Unity App Shop with a charge.

The current version has no more drawbacks of its own, but it could benefit greatly from implementation of more features. These features are discussed below.

- New models support: A new weapon model will be added and the suited damage system will be updated also. It is a large challenge to change the model of enemies. In the game, the 3rd character model was imported from the Character Controller which is the default package in Unity, thus the model is rough, and there are only four types of the animation. If the time will be adequate, the character model will be changed also.
- Updating system AI: Currently the AIs of enemies are hatred activated and patrol. The damage way depending on the distance between the player and the enemy is considered, which means the enemies make the higher damage if the distance is shorter. The way of the patrol which the enemies make is monotonous. The larger area which the enemies patrol could be implemented with waypoints, which means there would be fixed waypoints to make the enemies patrolling with the regular way.
- Mission Controller: The current way of healing the HP of the player is only heal-boxes. The heal-boxes are limited. If the player cannot slay all enemies before all heal-boxes are using up, the player is quite dangerous. Thus a new way of increasing the HP is that a new game object called NPC is created, and the

player can activate the mission from the NPC, when the player finishes the mission, he or she would get 100 HP. Also the mission is easy to be finished.

- More platforms could be supported: As we know, Unity supports the cross-platform, thus the game could be made on more game platforms. The iOS and Android are the best choice. Because the mobile game is the most popular nowadays.

6.2 Business and Marketing Strategies

There are several business models that can be used for the game. The first one is a free version model where the app could be downloaded for free from the App Store of Unity or Apple with free content. In this model the revenue will be coming from advertisements on the pages of the game. The advantage is the game can be updated slowly, and the disadvantage is the revenue is little. Another option is to make a premium version where users are entitled to all the contents for free once they purchase the game or a premium subscription account with a monthly or yearly fee. This requires the game must be updated fast. More new features can attract more customers.

A survey about the usability was distributed to ten players. The result is that half of them thought the playability of the game was significant game, and three people thought it was just a recreating game, and the rest of the players thought it was an excellent game. The feasibility for placing it on the market would not be large.

Fortunately, the App Store supports a large App platform for the designers. One's usages of apps on iPads, rather than iPhones or iPod Touches have sky rocketed in 2011. According to App Annie, downloads on iPad for 2011 have grown 200%, compared to 70% for downloads on iPhone. It is a reliable choice to the App Store. [12]

The truth is that a wonderful product occupies the market. A game that would occupy a seat in the market would require special features and attracting playability. Thus improving the game is the goal.

7 Conclusion

The aim of this project was to create a 3D game which was the First-Person Shooter game with the Unity game engine. This required I had a fundamental knowledge about the Unity game engine and programming. A game engine is the core of creating a game. The integration of model design, level design and script design is the game engine, which is complex and powerful. The Unity game engine supports visualized design, thus it is a strong game engine which is suitable for a beginner. However, it is not very easy to learn the Unity game engine well. There are various functions to be realized.

In order to make the game to be an integrated game, two scenes were designed. One was the game start scene and the other one was the game scene. As a result, the game start scene was achieved with 15 textures and pictures. Although just a few textures were used for the game scene, three scripts were created for it. The number of codes outnumbered 550 lines.

All of the game features were achieved as the First-Person Shooter game. More game features were looking forward to the future. The goal of the report was to demonstrate how to create a 3D game with the Unity game engine and discuss the implementation of the scripts. The damage system was one of the special designs. Two kinds of common damage ways were explained. The usages of the Skyboxes and the terrain design were the characteristics in the game.

The most important aspect of the game design is thinking how to create more new playabilities. This game is a product of a beginner who favours creating a virtual world. There might be more possibilities for the game in the future. This report demonstrated the whole process of making an FPS game with Unity.

References

1 Penny de Byl. Holistic Game Development with Unity. Focal Press: 1 edition November 15, 2011.

2 Xavier Borg. Understanding 2 dimensional spaces [Online]. Blaze labs research. URL: http://www.blazelabs.com/f-u-hds.asp . Accessed 10 September 2013.

3 Wikipedia. Three-dimensional space [Online]. Wikipedia, the free encyclopaedia.
URL: http://en.wikipedia.org/wiki/Three-dimensional_space. Accessed 11 September 2013. Accessed 11 September 2013

4 Patrick Congdon. Academic Word: Ten Maxims Every FPS Should Follow [Online]. Website: GameCareerGuid.com; 26 February 2007.
URL:
http://www.gamecareerguide.com/features/344/the_academic_word_ten_maxims_every_fps_should_follow.php?page=1. Accessed 12 September 2013.

5 Iuppa Nick, Borst Terry. End-to-end game development. USA: Elsevier, Inc.; 2010.

6 Sue Blackman. Beginning 3D Game Development with Unity: The World's Most Widely Used Multiplatform Game Engine. New York, USA: Springer Science + Business Media, LLC.; 2011.

7 Philip Chu. Game Development with Unity. [Online] Copyright © 2003-2010 Technicat, LLC.
URL: http://www.cocoachina.com/downloads/video/2010/0617/1698.html. Accessed 15 October 2013.

8 Webopedia. Game Level [Online]. ITBUSINESSEDGE, Property of Quinstreet Enterprise.
URL: http://www.webopedia.com/TERM/G/game_level.html . Accessed 10 November 2013.

9 Wikipedia. Level (Video Gaming) [Online]. Wikipedia, the free encyclopaedia. URL: http://en.wikipedia.org/wiki/Level_(video_gaming) . Accessed 15 November 2013.

10 Duion. Damage Systems [Online]. FreeGameDev Forums. 25 June 2013. URL: http://forum.freegamedev.net/viewtopic.php?f=5&t=4645 .Accessed 14 December 2013.

11 Michelle Menard. Game Development with Unity. USA: Cengage Learning; 2012.

12 App Annie. Infographic: The rise of the planet of the apps [Online]. San Francisco, CA:App Annie; 16 February 2012.
URL: http://www.appannie.com/blog/infographic-rise/. Accessed 13 January 2014.

13 Jussi Kasurinen & Kari Smolander. What Do Game Developers Test in Their Products? [PDF]. Department of Software Engineering and Information Management, Lappeenranta University of Technology. Accessed 6 March 2014.

14 Techopedia. Definition - What does First Person Shooter (FPS) mean? [Online]. Techopedia.com.
URL: http://www.techopedia.com/definition/241/first-person-shooter-fps. Accessed 23 February 2014.

A survey On Players' Responses towards the FPS Game with Unity

This survey is used to collect the responses from the players who have played this game. The purpose is testing the usability of the game. There are three parts on the survey. The part Ⅰis required that the player to fill in their information. The part Ⅱ is the questionnaire about the completeness of the game. The player must spend few minutes to play the game and make the scores for the functionalities. The part Ⅲ is the questionnaire that is used to evaluate the level of the playability. The player has to choose an option which he or she favours.

Part Ⅰ: General Information

1. What is your age?

A. 16 – 18 B. 19-24 C. 25-30 D. over 30

2. What is your gender?

A. Female B. Male

3. What is your degree?

A. Bachelor B. Master C. Doctor

4. What is your degree programme?

Part Ⅱ: Players' Attitudes towards the Completeness of the Game

1. Does this game can be played with First-Person Perspective?

A. 0 (No) B. 1 (Yes) C. 2 (Yes, perfect)

2. Does the game can be played with Shooting Mode (Sniper Rifle Mode)?

A. 0 (No) B. 1 (Yes) C. 2 (Yes, perfect)

3. Does the game possess the terrain (trees, mountains and grassland)?

A. 0 (No) B. 1 (Yes) C. 2 (Yes, perfect)

4. Could you see the sky and white clouds in the game?

A. 0 (No) B. 1 (Yes) C. 2 (Yes, perfect)

5. How do you think the AI of the enemies (hatred, patrol, damage)?

A. 0 (They did nothing) B. 1 (They performed with any bug) C.2 (perfect)

6. How do you think the damage system in the game?

A. 0 (It is difficult to win) B. 1 (Playable) C. 2 (Very balanced)

7. Do the Heal-boxes work?

A. 0 (No) B. 1 (Yes) C. 2 (Yes, perfect)

8. How do you think the GUI of the game start?

A. 0 (ugly) B. 1 (Good) C. 2 (Perfect)

Part Ⅲ: Players' Attitude towards the playability

There are three options to be supported to the player. You can choose an option which you trend to. The option which you chose is the evaluation of the playability. Please to choose the option depends on your own experience.

A. The game is quite easy to play. It is just used to make recreation in short time.

B. The game has good playability. It is not only used to make recreation, but also as the educational material. Because it is a completed game with the special functionalities.

C. The game has a potential to be developed with business. The application can be downloaded with charge.

Please give some suggestions on the game

THANK YOU VERY MUCH FOR YOUR RESPONSES!