


Zhang Yancan

Create an Endless Running Game in Unity

Bachelor's Thesis
Information Technology

May 2016

DESCRIPTION

		Date of the bachelor's thesis 2/Dec/2016
Author(s) Zhang Yancan	Degree programme and option Information Technology	
Name of the bachelor's thesis Create an Endless Running Game in Unity		
<p>The fundamental purpose of the study is to explore how to create a game with Unity3D game engine. Another aim is to get familiar with the basic processes of making a game. By the end of the study, all the research objectives were achieved.</p> <p>In this study, the researcher firstly studied the theoretical frameworks of game engine and mainly focused on the Unity3D game engine. Then the theoretical knowledge was applied into practice. The project conducted during the research is to generate an endless running game, which allows the players getting points by keep moving on the ground and colleting coins that appeared during the game. In addition, the players need to dodge the enemies and pay attention to the gaps emerged on the ground. The outcomes of the study have accomplished the research purposes. The game created is able to function well during the gameplay as the researcher expected. All functions have displayed in game.</p>		
Subject headings, (keywords) Unity3D, Endless running game, C#		
Pages 34	Language English	URN
Remarks, notes on appendices		
Tutor Reijo Vuohelainen	Bachelor's thesis assigned by Mikkeli University of Applied Sciences (change to a company name, if applicable)	

CONTENTS

1	INTRODUCTION.....	1
2	THEORETICAL OF BACKGROUND GAME DESIGN	2
2.1	Game strategy design.....	2
2.2	Game balance.....	3
2.3	Game engines today.....	4
2.4	Programming languages	6
2.5	Game basic rules.....	7
2.6	Game scene.....	7
2.7	GameObject	8
2.8	In-Game user interface	11
2.9	Animation state machine	12
2.10	Enemy (AI).....	14
3	PRACTICAL PART	16
3.1	Game Assets	16
3.2	Starting with the ground	18
3.3	Starting on the Character	22
3.4	In-game objects.....	27
3.5	High score records	28
3.6	Building the game.....	30
4	THE FINAL RESULTS.....	31
5	CONCLUSIONS.....	31
	REFERENCES.....	33

1 INTRODUCTION

In recent years, game industries have entered a stage of rapid development. More different types of games have appeared on a variety of new platforms, because relatively low cost and huge profits have motivated more people to get involved in this area. Earlier people only played games on consoles or computers until now people are more willing to play games on mobile phones. A variety of platforms brings more intensive competitiveness. On the other hand, in addition to playing the games, more people are willing to participate in the production process of the games.

Nowadays, there are many game engines allow people to create their own games in an easier and more convenient way, and Unity is one of the most suitable game engine for the beginner. Unity3D used to develop video games for multiple platforms, such as Web, PC, MAC, IOS and PS3 etc. On the other hand, Unity3D supports a variety of programming languages, which enables the programmers to program with their familiar language.

For this project, I will use the Unity engine to create an endless running game. During this project, developers will get familiar with most of the features of Unity engine, and use them as much as possible in our process. Since the game is not only made with game engine, I also need to explore the links between Unity and other game related software. The purpose of this project is to show the complete processes of an independent game.

In this report, I will start with basic theoretical framework of the game design. Then I will introduce the Unity engine. After that, I will describe the implementation levels of the project. The implementation contains full details of the game process. Finally, I will make a conclusion of this project involving the future development. Through every stage of learning and exploring in this project, it allows us to have a better image of the game production procedure, Moreover, the project helps to determine the difficulties which may be encountered in the production process and how to solve those problems.

2 THEORETICAL OF BACKGROUND GAME DESIGN

Game production often involves many elements. Since there are a variety of elements involved in the game production process, before starting to make this endless running game. It makes sense to explain each element contained in the process to have a basic understanding before building the game. Therefore, I will explain the following concepts which may appear in the game. They include game strategy design, game balance, game engines, programming languages and so on. Now we start with developing the game strategy.

2.1 Game strategy design

There are a lot of games which are not complicated, and their gameplay is not wise either. But after the game was released, it has been widely praised. Such games were available in the market in the early years and the contents of games are comparatively original. The valuable game idea is one of the reasons that these games are successful and indeed popular among the players.

Usually, game designers design games for the players' curiosity. Moreover, the game designers should consider other essential parts of the game, such as UI, character and game story design. Additionally, the game designers need to consider whether this design will motivate players to continue playing and keep playing, rather than just duplicate processes and felt tired of it soon. (Fischer 2012)

Different circumstances in the virtual world can bring players' satisfaction. In the virtual game world. The original intention of the players is to entertain, instead of searching for justice and balance. Moreover, players spend time on games in order to satisfy themselves and give them enough happiness. Therefore, based on these conditions, the game design can be very flexible (Fischer 2012.) In addition, the game designer must have mature thinking in the planning stages of the game. Otherwise, the ideas will bring unbalanced or fatal errors into the game.

2.2 Game balance

The goal of the game is to bring fun and entertainment to the players. Thus, gameplay and the degree of entertainment is crucial for a game. Therefore, the game designers are adjusting the game balance to make the game playable and entertaining. Reasonable level of design allows players to have fun during the game time. In addition, the game design must be balanced. Without balance the game will become messy and terrible. Also, this defect will be reflected in the process of the game. For the game designers, no balance means that a part of the game design may be reused many times, while some other part is rarely used. This will waste a lot of time spent on the game level design. A reasonable game should go through a series of choices and finally end with a victory or other results. For example, at a certain stage of the game, if it appeared with only one option that went to endless, the error of imbalance in the game occurred. (Francis, 2014)

Unbalanced games usually bring players a lot of troubles. It may not be obvious at the beginning of the game. However, while the game process goes further, the meaningless difficulty increases, which will lead to the players' declining interests towards the game. For example, if there is one character in the game, and he can always beat the other player's character at the same point under the conditions that both characters have full equality in skills and reflexes. Then, this game is clearly unbalanced. In most cases, the game designers' goal is to make reasonable arrangements for all attributes in the game, so that the victory can only be achieved through intelligent strategies and judgments. On the other hand, strategy often contains a lot of luck. Also, determining when to take risks is part of the fun of the game. At the same time, game designers must consider the balance of the game, to avoid the expansion of such factors during the whole process of the game, so that winning or losing is not decided by luck. (Francis 2014) As Figure 1 shows, for example.

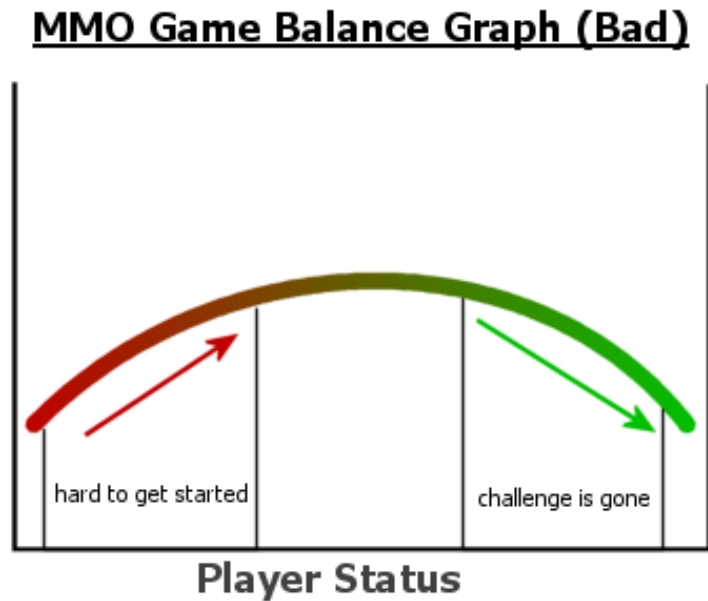


FIGURE 1. Bad game balance graph (MMO Bad Game Balance Graph)

Almost all the reasons for the imbalance of the game were caused by decreasing the player's choices. For example in a strategy game, if a unit's functions and costs have a great advantage compared to others, this will cause other components to become almost meaningless. This situation does not only left one option to the players, but also gives players a lot of irrelevant disturbances. These disturbances let the game become so confused, that it will reduce the gameplay and make game players feel bored.

2.3 Game engines today

More than ten years ago, the games were very simple and the capacity of the game was very small. Also, the game designers making each game often needed to rewrite all the codes. There was a lot of duplication of work and it was time-consuming. On the other hand, the programmers summed up a regular pattern. They used the generic code into the similar type of games. In this way, the programmers could largely reduce development costs and shorten the development cycle. Thus, these generic codes slowly became a prototype of the game engine. Finally, with the development of technology it evolved into today's game engine. Nowadays, game engines already developed from the early game accessories into a major role in game design. What kind of effect can be achieved in a game largely depends on how powerful of the engine is. (Masters 2015)

An excellent game engine usually contains several advantages. First, the game engine must have the complete game function. Nowadays, the game engine is no longer a simple 3D graphics engine. It covers for example 3D graphic, audio processing, AI operations, physical collisions and other components of the game. Also, an excellent game engine includes a powerful editor, including the scene, model animation etc. The more powerful editor, the more effects it can make into the game. Moreover, strong third-party plug-in support makes game development more smooth, such as 3Ds Max, Maya and other third-party software. In addition, the game engine needs to provide a simple and effective Software Development Kit, which helps game developers quickly get started. (Masters 2015)

In the current industry, the game engines for the mainstream of the game companies are Unity3D, Source 2, Unreal Engine and CryENGINE. Starting with Unity3D, it is a truly affordable engine for game developers; it has an unmatched number of users compared to other engines. More importantly, developers only need to pay once and no matter how successful their game became, they do not need to worry about Unity getting its share from the income. This is certainly an attractive feature, especially for the start-up companies and independent developers. Also, the learning threshold of Unity3D is very low and it is easy to use. Besides, Unity3D has strong developer community support and can be compatible with all game platforms. On the other hand, Unity3D has a limited number of tools, so developers need to create their own tools. It is much more time-consuming to make complex and diverse effects into the game.

Source engine are relatively backward compare with other engines. The graphic technology has fallen behind, but the source engine's light and physical systems are very good. The most powerful part is the original design intention of the source engine. It is very suitable for game players to take part into the game development. Also, source engine has a very powerful Software Development Kit. However, Source engine is only suitable for PC platform and hard to work with. The Source engine 2 has been announced, and it has a comprehensive upgrade compared with the previous version.

Unreal engine is the most popular game engine to make high-end games. The Unreal engine has high usage in game companies. Also, Unreal Engine has strong developer community support. There are a lot of video tutorials and resources available for users.

In addition, Unreal engine has the best engine support and updates. Each update will add new tools into the engine and management is relatively easy. But, the license terms only suitable for big companies, Unreal engine will get shares from the game when the game's income is over fifty thousand dollars. There are also some tools that are difficult to use and requires higher learning threshold.

CryENGINE gets a lot of recognition from game developers by the high quality of the game graphical capabilities. The art programming capabilities of Flowgraph tools are very powerful. Also, CryENGINE has the most powerful audio tools. These tools can bring the best sensory experience to the game players. In addition, CryENGINE provides the easiest UI code technology to the new game developers. But its developer community is relatively weak and required high learning threshold.

2.4 Programming languages

During the game process, one of the key parts of the game is to write the codes. Before the game programming, we need to choose a programming language to use for writing the code of the game. In today's game production environment, game developers have many language options to select them. Each language has its features, and the mainstream programming languages are JavaScript, C, C++, C#, Python, etc.

JavaScript has a very high demand on web applications. JavaScript is mainly used for web browsers to provide an enhanced user interface and dynamic websites. With the development of Node.js JavaScript has reached the mainstream of the server-side development. JavaScript is easy to use and it has similar syntax with Java. Also, JavaScript can be edited by using any text editor. JavaScript can execute the program that only requires a browser. On the other hand, JavaScript is not suitable for the development of large applications. (Hiscott 2014)

C and C++ are based on the C language. These are the most popular game programming languages. C is often being used for systems and applications, such as embedded system

applications. C++ is the enhanced version of the C language. It is used to develop system software, applications, device drivers, high-performance server and client applications and entertainment software, such as video games. (Hiscott 2014)

C# is an accurate, simple and object-oriented programming language which derived from C and C++. Also, C# inherits the powerful features of C and C++. At the same time, C# gets rid of some complex characteristics from C and C++. On the other hand, C# does not apply to write very high performance code, and lack of key features that high-performance applications required. (Hiscott, 2014)

Python is a dynamic language used to design a wide variety of applications. Python is often easier to write codes compare to other programming languages. Python's syntax is simple and clear; Python has a rich and powerful class library. Also, Python can easily link other languages modules together, especially C and C++. On the other hand, its performance is relatively poor. (Hiscott, 2014)

2.5 Game basic rules

In this game, we will control the character running from the left side of the screen to the right. The character will not stop running during the game unless the character is dead. The player can only control the character to jump to avoid obstacles or enemies. Also, the player has only one chance during the game. If the character dies, the game is over. During the process of the game, the system will record a score based on the running distance. If the character dies, the record will stop. When the next game begins, the previous highest score will be displayed on the screen. Furthermore, the character can increase the score by getting in-game items. Also, during the game making process, the in-game items may add some other features. Therefore, this game will show most of the basic contents of an endless running game.

2.6 Game scene

Before starting creating a game, we need to first set up an environment to make the game work properly. In the Unity3D engine, the scenes are responsible for providing such an environment. Scenes contain all the objects of the game. They also allow creating other parts of the game, such as the main menu, the different game levels and other

details. In addition, scenes can be used as an independent game level, so we can create different game levels by switching the scenes. In each scene, developers can put their characters, buildings and obstacles then build their game (Unity Documentation 2016).

2.7 GameObject

In the process of making the game, the GameObject is one of the most important objects that we will use in Unity. The definition of GameObject is a little bit vague. It could be any object in the game that the player can interact with. Before GameObject becomes a specific thing, we need to give the object some special properties, so that the GameObject can become different pieces in the game such as a man, a building, or something that players may use in the game scene. Therefore, we can consider it as a container. When we need an object to become a specific character or environment, we need to add different properties into GameObject (Unity Documentation 2016). Moreover, the GameObject typically contains some basic properties, and each property will have different effects on the GameObject. GameObject can also affect the significance of the object in the game. Therefore, understanding each property is very important.

2.7.1 Transform

The Transform functions of GameObject in Unity are to control its physical parameters. Transform is used to control GameObject's position in the game. The Rotate fromation is used to control GameObject's orientation. The Scale is used to control GameObject's size in the game. The Transform is manipulated in 3D space on the X, Y and Z axes, but in 2D space only on the X and Y axes, which is shown in Figure 2.

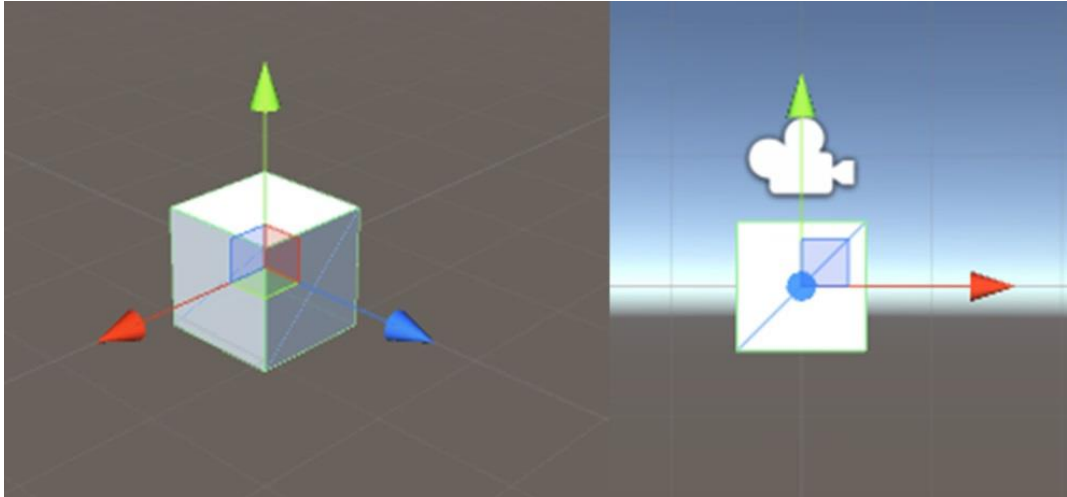


FIGURE 2. X, Y and Z axes in 3D and 2D space

The Transform functions are usually configured by scripting, so that Transform can be effective in the game. In different programming languages, the scripting form can be quite different. In this project, we will use C# to complete this game. Therefore, when configuring GameObject in C#, the basic form within a script is shown in figure 3.

```
gameObject.transform.position = vector3(x, y, z);
gameObject.transform.Rotate(new Vector3(x, y, z));
gameObject.transform.localScale = new Vector3(x, y, z);
```

FIGURE 3. The basic script form of the GameObject Transform

In Figure 3, the position and scale are stored as vector3 because GameObject is in the 3D environment, which means that we can configure the X, Y and Z axes. On the other hand, in 2D environment we can only use the X and Y axes.

2.7.1.1 Transform.parent

In Unity, parenting is one of the most important concepts during the game production process. For example, if GameObject is a parent of another GameObject, its Child GameObject will make the same changes of its transform like its Parent does (Unity Documentation 2016.) In game, transform.parent can be used for binding the GameObjects or camera with the character or other GameObjects. The transform.parent is basically script as Figure 4 shows.

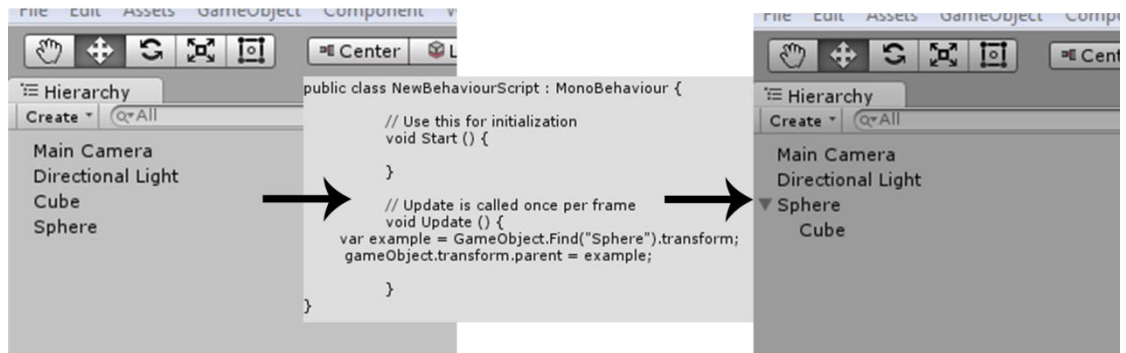


FIGURE 4. Work process of the Transform.parent

Also, transform.parent can be done by only dragging a GameObject onto another one. But in most cases, we will choose to use the script method, since it can cause some unpredictable problems.

2.7.2 Collider

In the game, the collisions were often used in the game objects. Therefore, we need to create a medium for the game objects which is called Collider. In addition, Collider is invisible and it does not need to have the same shape like the mesh of the GameObject. In 3D game, there are Box Collider, Sphere Collider, Capsule Collider, Mesh Collider, Wheel Collider and Terrain Collider. In 2D game, there are Box Collider 2D and Circle Collider 2D. 3D colliders as in Figure 5.

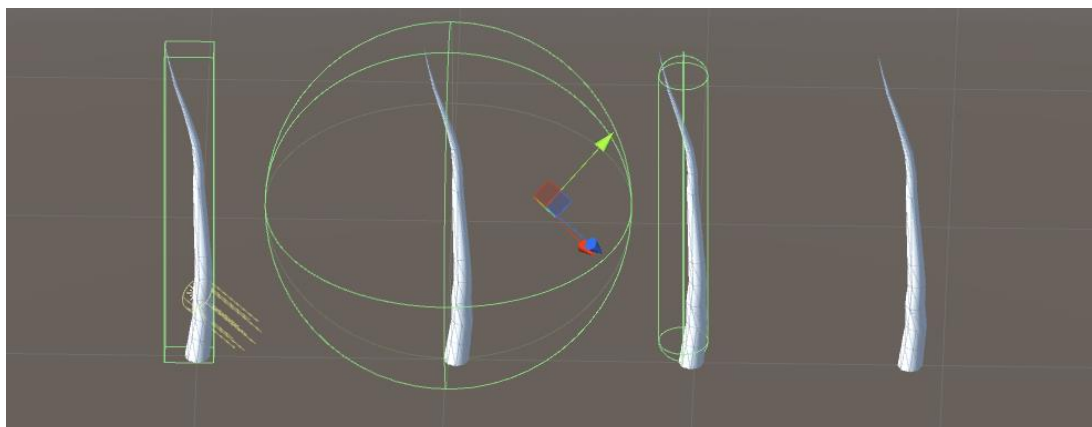


FIGURE 5. Capsule Collider, Mesh Collider, Wheel Collider and Terrain Collider

In most cases, the Box Collider and the Sphere Collider are the most efficient colliders. Also, less shape and size of a game object can keep lower processor overhead. For example, buildings and basic items in the game do not have many shapes to make the processor calculate, in this case, the Box collider could be used for these GameObjects. Moreover, when Box Collider and the Sphere Collider are not enough to fit the GameObject, we can use Mesh Collider to match the mesh of the GameObject. Mesh Collider needs consume more processor performance than the primitive colliders, so that we should not use this Collider frequently.

In some cases, the Colliders can be set without a Rigidbody, like when creating the floors, building walls and static items in our game. Therefore, we normally do not reposition these Static Colliders with the Transform position, because these Static Colliders will affect the performance of the game engine. These GameObjects with Rigidbody can interact with the Static Colliders but the Static Colliders will not response to any collisions since they do not have the Rigidbody. (Unity Documentation 2016)

2.7.3 Materials, Shaders and Textures

When put the character into a game, usually the designer needs to add materials, shaders and textures in order to make the character looks more realistic. First of all, the material defines how the surface should be rendered, and the rendering option is normally depending on which shader that the material is going to use. Then, the shader calculates the color rendered form the material. Finally, the texture is used to bitmapping the images to the GameObject (Unity Documentation 2016.) In most cases, we use the Standard Shader to the GameObjects. As a new type of the Built-in shader, it can handle most of situations. Also, the Standard Shader can help developers reduce the time spends on selecting a shader from the shader lists.

2.8 In-Game user interface

When we play a game, game UI is related to all the interactions we made during the game. Also, game UI refers to the interaction between human and computer, the operating logic and the overall beautiful design of game interface. In addition, a good UI design is not only make the product fits personal preferences, but also make the production became more comfortable, simple, free and fully reflect the characteristics of the

game. Normally, a game UI includes the game menu, keyboard control, mouse control, in-game icons and all the elements players could be interacted in the game. As figure 6 shows.



FIGURE 6. Example of Game UI (Source: Mobile Game UI. Image by Alex Parker)

In Unity3D, we have two GUI options to choose, they are UnityGUI and Next-Gen UI. The UnityGUI is an officially build-in GUI system and the Next-Gen UI is a plugin for Unity3D. In many games, game designers are preferring to use NGUI to complete the UI system, because the UnityGUI is much more sophisticated in the operating phase. In addition, developers need to script the entire UI for the labels, textures and other UI elements in game. Moreover, the Next-Gen UI is easier to use since the UI elements of Next-Gen UI are the GameObjects in Unity3D. In Next-Gen UI you can easily edit your labels by creating the widgets, which are viewed by a camera so that you can check how the UI looks from a game window. (Charles 2014)

2.9 Animation state machine

Game character is one of the most important parts in a game, since the influence of the character animation is a significant in game (Unity Documentation 2016.) Also, interaction between the game characters and the game objects should be adjusted in the An-

imation State Machine. In addition, animation state is frequent used in game. In addition, not only the role-playing game using the Animation State machine, but also be used in place if any object is switching action in game, as Figure 7 shows.

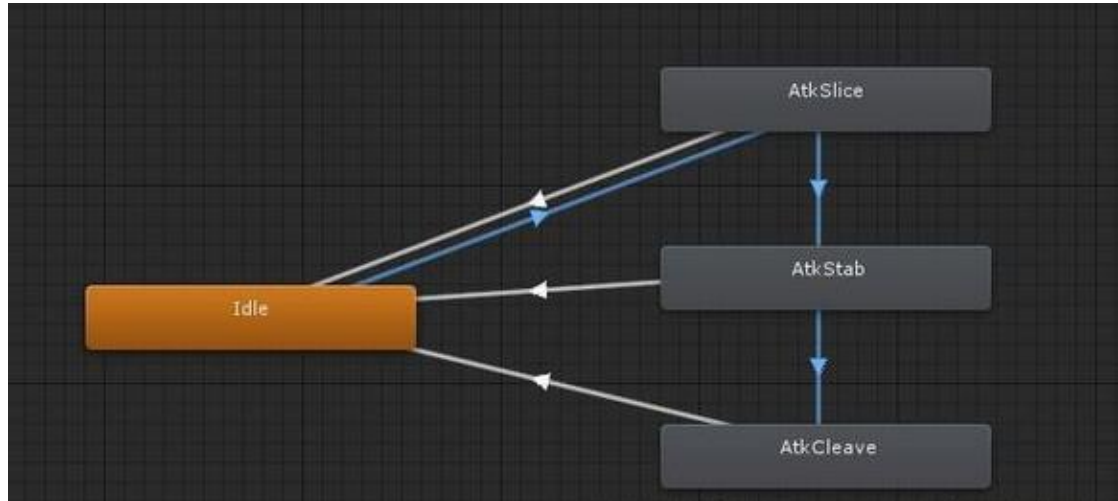


FIGURE 7. Animation state machine in Unity3D

There are some main points of animation states in game. First of all, game character usually has 4 to 5 basic actions, such as idling, walking, attacking, being injured and so on. All of these actions are called as states. We need to connecting all these states with a parameter into the Animation State Machine. Then, we need to give the basic animation states to enemies. In Unity3D, using the Animation State Machine to editing the AI of enemies is the fastest and most effective way. Usually enemies will have some extra states, such as chasing, fleeing, dead and so on. Finally, we need to consider about character instruction to support the character action states. We found that there are only the states when the character action states were finished in Animation State Machine, but we also need to editing some extend details for example Transform or Scale to character. The character instruction state has the responsibility to these situations. It is another layer over the character action state. For example, the walking instruction is using the walking animation state and keeping the game character in walking state. Then we only need to deal with the Transform for game character. Now we had both action and Transform, so that the entire movement instruction is complete.

2.10 Enemy (AI)

In most games, AI is always an integral part. An excellent AI design can provide an enjoyable game experience for players, and make the visual effects and animation be more interesting. Since the main purpose of AI is to create the gaming experience for players, therefore, AI must be supporting the overall gaming experience instead of just showing how smart of the AI we made. When creating an excellent AI, the most important thing is what kind of gaming experience it brought to the game and what problems it can helping us in game. On the other hand, poor AI is considered as a serious threat in game, which can ruin the game experience that we are trying to build. And it also gives a devastating impact to the game from the beginning to the end. (Wenderlich 2012)

Game AI could be simply understood as the character that controlled by an intelligent computer. These intelligent characters can determine any changes from the surrounding environment or events, so that game AI can produce a specific behavior from player's action. There are some basic AI elements for example the basic logic of AI, AI basic skills, and basic properties of AI etc.

The Basic logic of AI can be divided into the perception, the action and the reaction. First, the perception is the capacities how AI determining changes from the surrounding environment which is given by the game designer. For example, in stealth games, the vision of enemies only has a 90-degree angle wide arc in front of them. If the set the AI only has this perspective, then enemies will only react when player is in this view field. Then, AI will be decided to do a series of behaviors, these AI actions are set from a series of rules and logical order by game designer (Wenderlich, 2012.) For example, if game designer wants to create a vibrant city, we can create many different AI characters, and some of them may running from one place to another place along the street, some of them may chatting in a plaza etc. In some survivor games, zombies will wait for an opportunity to attack the player after detecting player's position. If the player shot zombies, they will try to dodge.

When creating AI basic skills, the game designer must first analyze basic abilities of the game AI, which can help game designer to make extension abilities much easier. Also in most of games, the game AI ability generally can have for example detecting potential

threats and confirming other's identity (friend or enemy). Furthermore, we can use these basic capabilities as the parameters to designing more different types of AI. For example, enemy can detect to attack, run or dead, also can detect to use skills, attack, run or dead.

Basic properties design for a AI is extremely important, since it will affect the game balance. Most game AI basic properties can be divided into four parts, including identity, combat parameter (the value of life, attack and defense), interaction range (distance of chase, hatred detection range, attack distance) and aggressive behavior (the frequencies of attack skills). Game designers can adjust the basic properties and design different personality AI. For example, the properties of a warrior might be high life value, normal attack value and slow speed, but an assassin's properties could be low life value, high attack value and high speed. The differences between them are the appearance, animation, life value and speed. But they have the same AI logic.

It is very important to design game AI with these three elements. First, we need to set the basic definition of AI challenge in game. Single AI challenge often requires a combination of the challenging level design to consist a basic challenge pattern. Before game designer get into the detailed conceptual design, they must first understand where is the main gameplay part in game, which part of game will have a game AI, and what kind of character we are playing. Secondly, we need to design the basic skills and basic properties of AI in the challenge pattern. Once a combination of diverse abilities and attributes have done, we can explore more diverse challenges. Then, we need to design the AI logic diagram, because a clearly structured game AI running process will help programmers to understand the behavior of game AI better. As Figure 8 shows, for example.

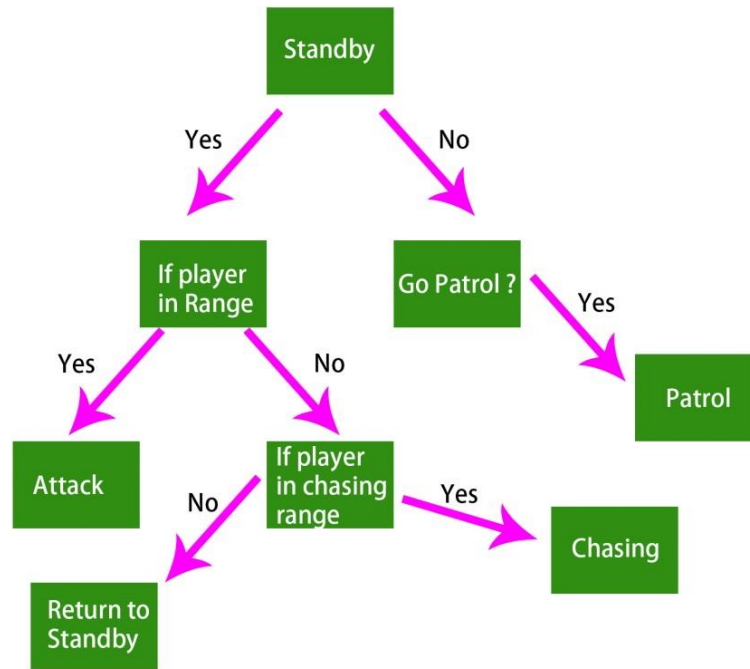


FIGURE 8. The example of AI logic diagram

It can also be understood as a state machine, AI will complete a specific behavior in one state, when specified conditions are met, and the AI will move from one state to another state. Therefore, the AI logic is switching back and forth between AI state conditions.

3 PRACTICAL PART

In the previous chapters, we discussed the basic theory background of the game balance, programming with Unity and Unity features. As the final goal of the project, I need to create an endless running game which covers the complete features. Therefore, this chapter will discuss the implementation stage of the game creation process. For the basic game rules, the player will run from left to right in the game scene. There will be some obstacles in the game. Players need to jump through these obstacles, otherwise the game will end.

3.1 Game Assets

The game's visual experiences to the players are often achieved through the game assets. The game assets are often divided into art assets and script assets. The game art

assets refer to the game models, model textures, game background music and etc. These art assets make up the in-game objects and user interfaces. Instead, scripts assets are the core parts of the game. These assets are programmed with the code which is responsible for the administration of the gameplay and rules of the game.

During this game project, images are designed to create the user interface, characters and scene items. Since this game project is a 2D side scroller game, we do not need any 3D game model in the game. All characters and in-game items (for example, ground, background, obstacle etc.) are designed and created by Photoshop during the project. As shown in Figure 9.

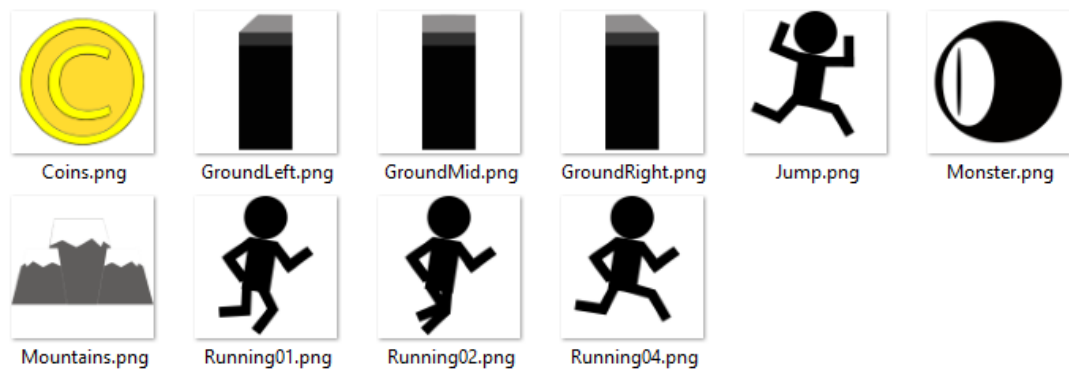


FIGURE 9. Art assets for in-game items

For the character, I made multiple sets of images for the character's action such as jumping and running. Each single image represents one frame of the action. The animation will be produced by looping the image set from the first to the last image. The entire action image set will be adjusted in the animation state machine. Also, I need to adjust the timeline of each image in the animation state machine to make the action become natural.

The ground is divided into three parts, left, right and middle, since the ground is randomly generated from the right to the left in the process of the game. In addition, there will be many gaps between grounds. That is why the ground must have beginnings and ends.

The background is divided into two parts. The sky will be created by a single static image. On the other hand, the mountains and clouds will be dynamically generated like the ground, also from the right to the left side. But the moving speed of the mountains and clouds is slower than the ground. Also, clouds and mountains have different speed.

Moreover, the script assets are essential to the game project as well. The scripts are the key to make the gameplay and performance in the right way. During this project, I will use C# to create the scripts.

3.2 Starting with the ground

Starting the game stage, we first need to create the basic ground to let the main character walk on it. First, I need to edit these art assets of the grounds, so that it can fit into the game. One of the most important things is to add a 2D collider to the ground, so that the ground can contact with the main character. Also, the 2D collider can let the character stand on the art assets in the physical environment. In this project, all the settings of the game asset are edited in the inspector section. This section allows the user to change the initial values of the object, which contains the basic values such as Transform and Renderer, and we can add new components later. Also, the inspector allows the user to change the values in the game operating mode. The ground assets do not need editing its transform, but for the 2D collider I have changed its offset and the collider size to let the collider fit into the game assets, as figure 10 shows.

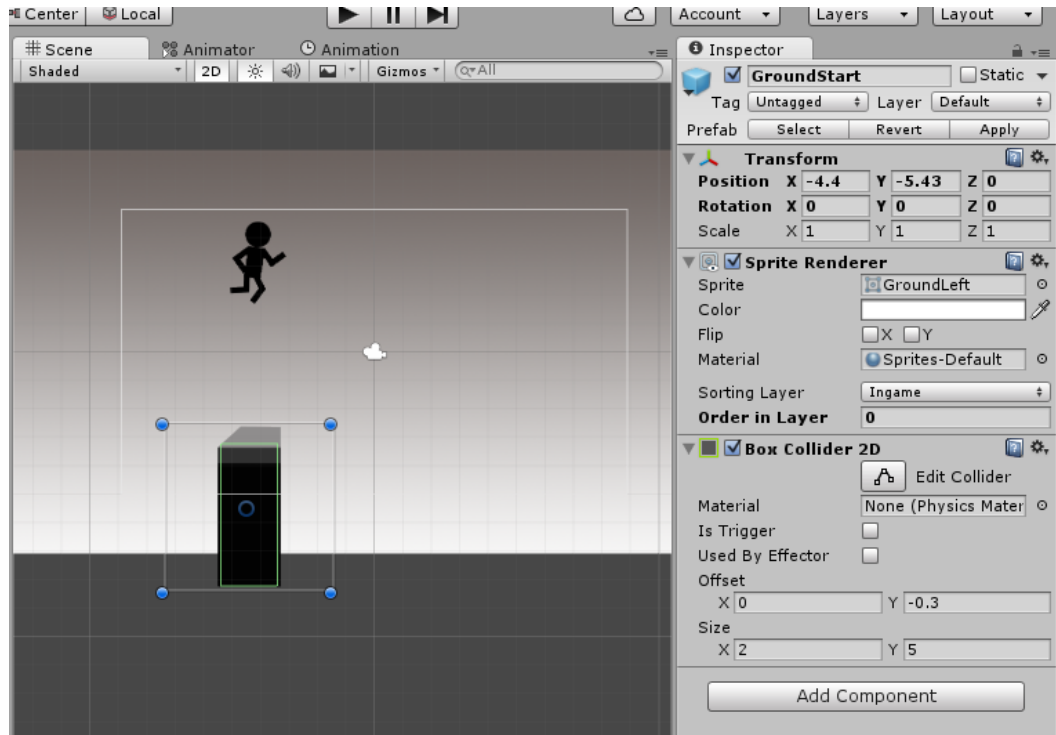


FIGURE 10. The basic settings of the ground prefab

Also, we need to create the prefabs of the grounds for later use. The main object of using prefabs is to allow the game objects and resources to be reused during the game process. Prefabs can improve resource utilization and efficiency of the development in this project. It is important to create the prefabs for each of them. Also in the hierarchy section, I create some empty game objects called GroundsLayer, GameLayer and BgLayer. These empty game objects can help me to collect these instantiated prefabs during the game process. This way I can also make the hierarchy more concise and specific. During the game process, each ground prefab will be generating under the sublayer of the GroundsLayer, as in Figure 11.



FIGURE 11. Prefabs of the right ground collected by the empty game object called gRight

Furthermore, we need to create the scripts to implement the process of generating the grounds. In the script, I create few codes to find the prefabs under the Resources folder in the Project section. Then, is instantiate the prefabs of the grounds and collect them into the GroundsLayer. In addition, I create a game object called GroundStart, which is the same as the prefab of GroundLeft, so that all the grounds can instantiate behind the GourndStart to make a complete road. Code 1 shows this process.

```

for (int i = 0; i < 20; i++)
{
    GameObject temg1 = Instantiate (Resources.Load ("GroundLeft", typeof(GameObject))) as GameObject;
    temg1.transform.parent = collectedGrounds.transform.FindChild ("gLeft").transform;
    GameObject temg2 = Instantiate (Resources.Load ("GroundMid", typeof(GameObject))) as GameObject;
    temg2.transform.parent = collectedGrounds.transform.FindChild ("gMiddle").transform;
    GameObject temg3 = Instantiate (Resources.Load ("GroundRight", typeof(GameObject))) as GameObject;
    temg3.transform.parent = collectedGrounds.transform.FindChild ("gRight").transform;
    GameObject temg4 = Instantiate (Resources.Load ("GroundBlank", typeof(GameObject))) as GameObject;
    temg4.transform.parent = collectedGrounds.transform.FindChild ("gBlank").transform;
}

collectedGrounds.transform.position = new Vector2(-60.0f, -20.0f);

GroundPosition = GameObject.Find("GroundStart");
startUpPositionY = GroundPosition.transform.position.y;

```

CODE 1. Generating the prefabs from the Resources folder.

At this point, the ground assets can be made into a complete road. Each of the ground with 2D collider can allow the character to run on it. In all the endless running games, the game scene often moves in a horizontal or vertical way. With the movement of the scene, the level of the ground will change frequently. In this game, the ground will change the height and width randomly. Also, the width of the gap between the grounds are randomly different.

First, the grounds and backgrounds need to be in motion. Since the game is in two-dimensional perspective, we only need to configure the axes X and Y. Also, the game is scrolled in horizontal movement and then we should only configure the Transform position of X axes. Moreover, the grounds and backgrounds are edited with different game speed in real time. The game speed of the backgrounds is much slower than the ground, which make the game seems more realistic. In addition, the default game speed is set up manually, as Code 2 shows.

```
gameLayer.transform.position = new Vector2 (gameLayer.transform.position.x - gameSpeed * Time.deltaTime, 0);  
bgLayer.transform.position = new Vector2 (bgLayer.transform.position.x - gameSpeed / 5 * Time.deltaTime, 0);
```

CODE 2. Movement of the grounds and backgrounds

After completing the movement of the grounds. We need to edit the height level of the grounds. In the process of the game, the width and height of the ground are generated randomly, and the gap between the grounds is not the same. Furthermore, the random range of the grounds and gap need to be designed within a range, since oversized width of the grounds and the gap can affect the game balance. The oversized ground will make the game boring and the game cannot be continued. Therefore, we set the range of the grounds and gap as in Code 3:


```

private void generateGround()
{
    if (blankCounter > 0)
    {
        setGround("blank");
        blankCounter--;
        return;
    }
    if (middleCounter > 0)
    {
        setGround("middle");
        middleCounter--;
        return;
    }
    if (lastGround == "blank") {
        changeHeight ();
        setGround ("left");
        middleCounter = (int)Random.Range (2, 10);
    }
    else if (lastGround == "right")
    {
        blankCounter = (int)Random.Range (2, 4);
    }
    else if (lastGround == "middle")
    {
        setGround ("right");
    }
}
}

```

CODE 3. Generating the grounds and setting them randomly

All the situation of the grounds is shown in code 3, so that if the blank grounds or middle grounds are over 0, the middle grounds reduce its counts. Also, if the last ground is the blank ground, the scene should generate a new ground, or a new start. The middle ground will be generated during the process, and the number of the middle grounds should be in a random range, so that the game will not look so boring. Moreover, if the last ground is the right ground, it means that the blank ground will be the next generated ground. Then, the scene needs to set the counts of the blank ground in a random range.

3.3 Starting on the Character

Another important element of the game is the main character. The players will control the game character to reach higher challenge during the game time. In this project, the character needs to add a set of animation, so that it will moves in motion. First, we need to create and put the character into an editor called the Animation. The editor allows the designer to edit the animation of the character by frames. During the game process, the running animation of the character is composed of three images. Additionally, a coherent animation requires the first frame and the last frame to be exactly the same to make the animation loop look more natural. As in Figure 10.



FIGURE 10. The animation of running in frames

In addition, the character needs some components to be added to be able to contact with the grounds. The first thing is to add the Box collider 2D, so that the character can have collision contacts with the grounds. On the other hand, since the default size of the collider is too large, and it can affect the game play, the size of the character needs to be reset to fit into the game. Also, it is important to give the character another component called Rigid body 2D. This component can give the character a physical effect such as mass, gravity and so on. Moreover, the rotation of the Z axis of the character in Constraints should be frozen, so that the character will not shake in the game process. This setting can prevent the character from not rotating during the game, as in Figure 11.

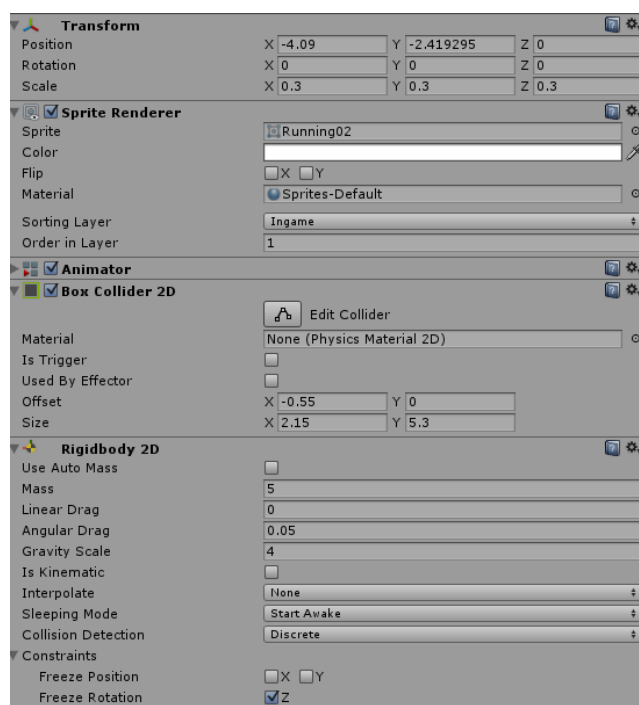


FIGURE 11. The properties of the character

After the animation and properties of the character are done. We need to add control to the character with some scripts. The basic control in this game is to right click the mouse, so that the character will jump over the gap to the other grounds. Also, in this game the character can only jump once at a time, so that there are two situations in the script, mouse button up and down, as code 4 shows the control part.

```
// Update is called once per frame
void Update () {
    if (Input.GetMouseButtonDown (0))
    {
        handleInteraction (true);
    }
    if (Input.GetMouseButtonUp (0))
    {
        handleInteraction (false);
    }
}

void handleInteraction(bool go)
{
    if (go) {
        player.jump ();
    } else {
        player.jumpHit = false;
    }
}
```

CODE 4. The basic control

Since the character will run and jump during the game process, the animation needs to switch between these two actions. Therefore, we need to give a parameter to the character in the Animator, like in Figure 12. First, the jump animation needs to be created in the Animation editor for the character. Then, running and jumping animations need to be connected to each other in the Animator by the transition components. Afterwards, all the transitions need to be added a parameter in the conditions setting called AnimateState, so that when the character takes an action and reaches the condition, the animation will switch to another one.

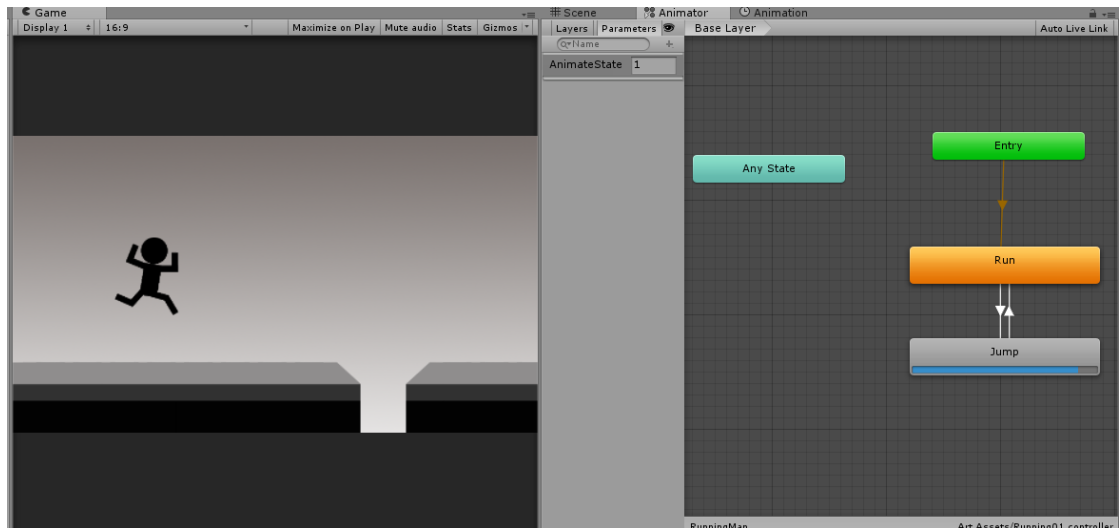


FIGURE 12. Character animation with the Animator

On the other hand, we also should give more details of the conditions with codes, so that the character can be allowed to switch the animation in the game. The animations are divide into two judgments in the codes. If the Y axis of the character is over the setting value, the character is determined as over the grounds. Therefore, the parameter is set to 1 in the conditions to play the jump animation. Then, if the Y axis of the character is equal to the setting value, the parameter is set to 0 to play the running animation, as in Code 5.

```
// Use this for initialization
void Start () {
    TheAnimator = GetComponent<Animator> ();
}

// Update is called once per frame
void Update () {
    if(!OverGrounds && theRG.velocity.y > 0.05f)
    {
        TheAnimator.SetInteger ("AnimateState", 1);
        OverGrounds = true;
    }
    else if(OverGrounds && theRG.velocity.y == 0.00f)
    {
        TheAnimator.SetInteger ("AnimateState", 0);
        OverGrounds = false;
        if (jumpHit)
            jump ();
    }
}
}
```

CODE 5. Switching between the animations with codes

Also, the character will be in contact with the enemies within the game. Like in most games, when the character contact with the monster, the character will trigger the death animation. In this project, this animation will be achieved through the codes shown in Code 6.

```
void OnTriggerEnter2D(Collider2D coll)
{
    if (coll.gameObject.tag == "RunningMan")
    {
        GameObject tmpPlayer = GameObject.Find ("RunningMan");

        tmpPlayer.GetComponent<Rigidbody2D> ().AddForce (Vector2.up * 2000);
        tmpPlayer.GetComponent<Collider2D> ().enabled = false;
    }
}
```

CODE 6. Codes for death animation

When the character is dead, the character will jump up with a force value. Also, the character will drop through the ground like in Figure 13. In the code the character will turn off its collider component to be able to pass through the colliders of the grounds.



FIGURE 13. The scene of death

3.4 In-game objects

In the game, the character does not just jump through the gap to reach the other grounds. There are also enemies. During the game process, the character also needs to avoid the monsters on the road. The monster has the basic components such as `Renderer` and `Collider2D`. In addition, `Collider 2d` is set as a trigger so that the monster has the trigger events with the character. The trigger event code is in Code 6. When the character contacts the monster, the trigger will be activated.

On the other hand, the monsters need to dynamically be generated in the game. The monsters should on the grounds and move together with the grounds. Also, only one monster will be generated in the platform in order to avoid the game being too difficult for players, as in Figure 14.

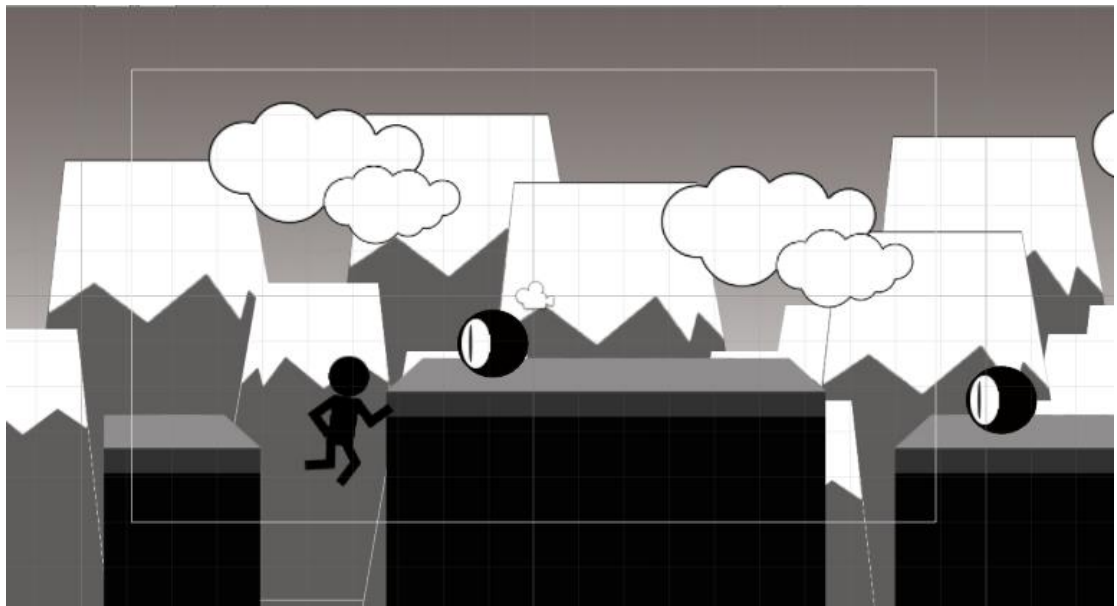


FIGURE 14. Always one monster in the platform (the white box)

The position of the monster on the grounds is generated randomly with the codes. Moreover, the monsters are always generating on the middle grounds to protect the game balance. Since if the monsters are generating on the first or the last ground, then the character cannot pass through the monster to reach the other side.

Another very important in-game item is the reward coins. The character is not just earning points by moving forward to reach other grounds. Also, the reward coin can give

very good bonus points to reach a high score. The reward coins have some basic principles compared to the monster. But the coin will disappear when the character collected it and then it will be respawning from the right side. These features are scripted with code in Code 7.

```

        if (!inPlay && !genCoins)
            respawn ();
    }

    void OnTriggerEnter2D(Collider2D coll)
    {
        if (coll.gameObject.tag == "RunningMan")
        {
            GameObject.Find ("Main Camera").GetComponent<ScoreHandler> ().Points += 10;
        }

        inPlay = false;
        this.transform.position = new Vector2 (this.transform.position.x, this.transform.
    }

    void respawn()
    {
        genCoins = true;
        Invoke ("placeCoins", (float)Random.Range (3, 10));
    }

```

CODE 7. generate the coins and the reward feature

Therefore, players need to control the character to get the coins as much as they can. At the same time, they also need to dodge the monsters to avoid death. All these in-game objects can make the game more flexible and fun.

3.5 High score records

In this game, the high score is to measure the distance that the player has reached so far. Also, the coins give bonus points. When the player reaches a new higher score, the high score record database will replace the old with the new score, as in Code 8.

```

    public void sendToHighScore()
    {
        if (TheScore > TheBest)
        {
            saveValue (TheScore);
        }
    }

```

CODE 8. Replacing the old score to a new high score

In this game, it is necessary to use GUI to display the score and high score on the platform. The GUI is implemented with codes and the basic interface should also be edited in the script. In Code 9, there are two parts of the score, including the current score and the high score. The current score is to tracking the current score that player achieved during the game process. The high score is to show the highest score so far.

```
void OnGUI()
{
    GUI.color = Color.black;
    GUIStyle Gstyle = GUI.skin.GetStyle ("Lable");
    Gstyle.alignment = TextAnchor.UpperLeft;
    Gstyle.fontSize = 20;
    GUI.Label (new Rect (20, 20, 200, 200), "Current: " + TheScore.ToString (), Gstyle);
    Gstyle.alignment = TextAnchor.UpperRight;
    GUI.Label (new Rect (Screen.width - 220, 20, 200, 200), "Highscore: " + TheBest.ToString (), Gstyle);
}
```

CODE 9. GUI settings in script

When the game starts, the current score will start counting the points that the character achieved. Also, the high score record will keep the previous high score until a new one replaces it. It looks like in Figure 15.

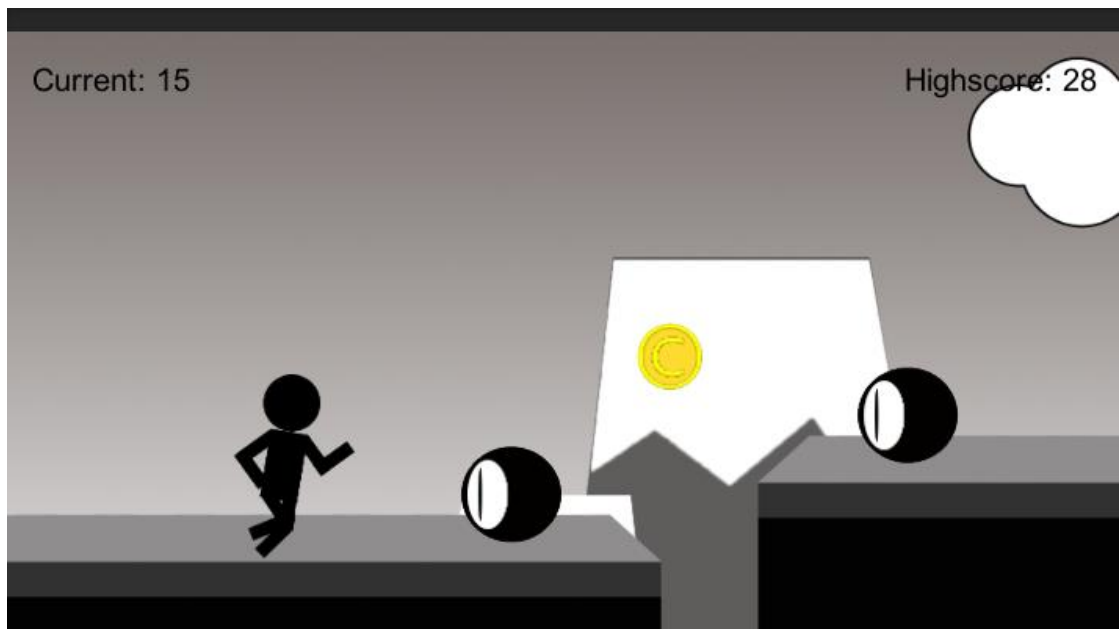


FIGURE 15. GUI of the current score and high score

3.6 Building the game

When all the scripts and in-game objects are ready to test, we only need to finish by clicking the Build & Settings button in the File tools, as in Figure 16. Since Unity 3D is a cross-platform engine, Unity 3D supports almost every game platform in the game market.

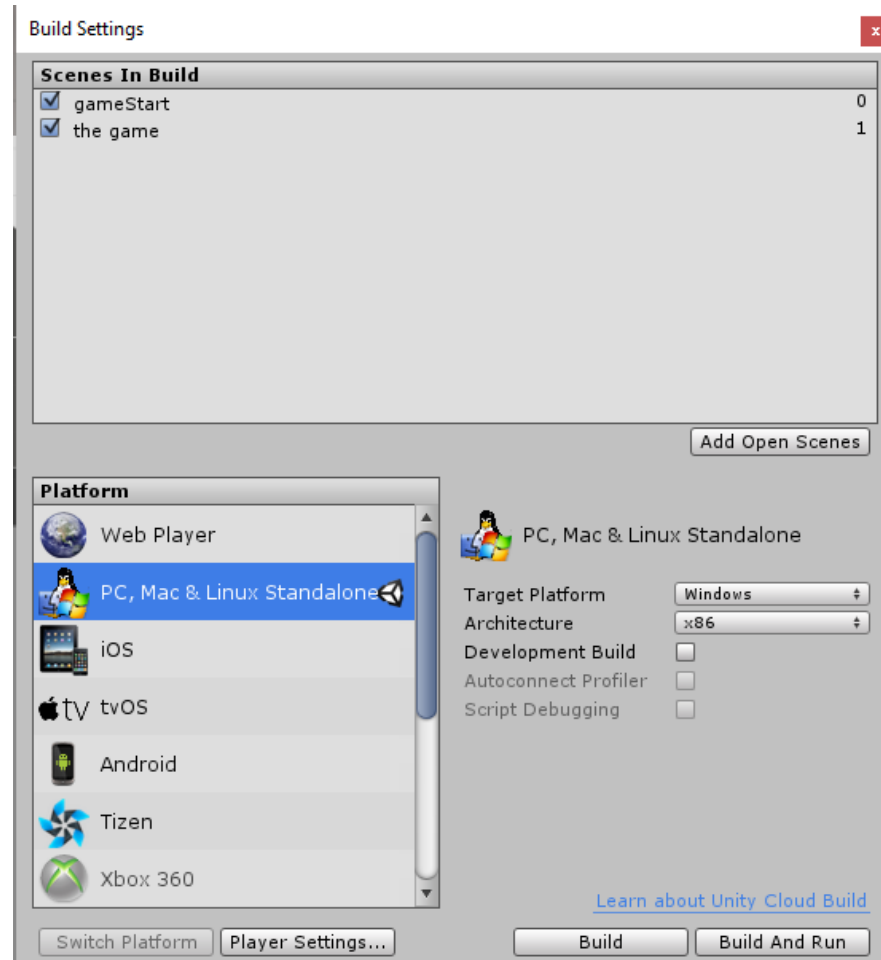


FIGURE 16. Build & Settings tool in Unity3D

Figure 16 shows the game scenes in the Scenes in Build window which includes the game start menu and the main game scene. Also, I choose to run this game on a PC platform, since another platform might need some advanced configurations. When all the settings are ready to run, we only have to click the Build or Build and Run button to make an exe file and play it.

4 THE FINAL RESULTS

During this project, all the basic features were achieved completely endless running game on a PC platform. All the game assets were created by myself with Photoshop software. In addition, all the assets in the game have some components to contact with other objects in the game. For the game play, the game player can control the character to jump and reach other grounds by right-clicking the mouse. The game points are increasing when the game goes on. Also, there are coins that can give bonus points and the monsters on the grounds that the player should dodge. The character dies, if he touches the monster. Then the score will be settled. If the new score is higher than the score record, the previous high score will be replaced by the new one. The main goal in this game is to get more points and break the old high score records. However, the way of running this game out of the Unity 3D engine is to run the exe file which is created from the Build & Setting tool.

5 CONCLUSIONS

The aim of the study was to create an 2D endless running game with the Unity 3D game engine. During this project, I gained a deeper understanding of the features of the Unity3D game engine and fundamental knowledge of game programming. Also, the game engine is an important tool for the game designer to start building a game. The Unity 3D game engine is a powerful game engine and suitable for beginners. But, to become a better game designer, there are many more functions of this game engine to be discovered. The theory backgrounds helped to understand the environment of programming and game engines. The game designer will become more familiar with the work environment. In this game, there are two scenes when the game starts. The game start scene is the in-game main menu. The player will switch to the gameplay scene from this scene.

During this project, every step of building this game was fairly successful. The game-play is working very smoothly. The in-game objects are doing their duties properly during the game process. On the other hand, there are still many features that can be added into this game; more in-game objects and game level design are necessary to make the game more interesting. Also, the current in-game objects and the main character can

add more features, such as more frames for the animation of the character to make it look more natural. This project still has a very large development space. The main objective of this project was to demonstrate the process of create the 2D game with the Unity 3D game engine.

There were few problems and difficulties during the implementation of the project, such as the style of programming in the new version of the Unity 3D game engine which had some changes. The game assets also involved some small issues. However, these problems and difficulties were solved during the project. By going through this project, I have improved my knowledge and skill of working with the Unity 3D game engine. Also, I consolidate my knowledge of programming with the C# language.

For further development, there are still a lot of elements can be added into this game. Lack of elements can make players lose interest fast. There are many details not taken into consideration. None optimization was made for the code in this project. All the work was done only based on what I had learned from the websites and books. With more in-depth study of programming with the C# language and the environment of the Unity 3D game engine, this game could be better.

The most important thing of designing this game is to discover how to develop a game and imporve it. There might be more things to add into this project in the further development. During this project I have learned how to work independently and solve all the problems and difficulties on my own. This project illustrated the whole process of making an endless running game with the Unity 3D game engine and C# language.

REFERENCES

Fischer, Fabian. 2014. Criteria for Strategy Game Design. WWW-doucement. http://www.gamasutra.com/blogs/FabianFischer/20141201/231243/Criteria_for_Strategy_Game_Design.php. Updated 12.1.2014. Referred 10.4. 2016.

Francis, Chris. 2014. The important of Gameplay Balance. WWW-doucement. <http://www.theoryofgaming.com/importance-gameplay-balance/>. Updated 14.5.2014. Referred 16.4. 2016.

Masters, Mark. 2014. Unity, Source 2, Unreal Engine 4, or CryENGINE - Which Game Engine Should I Choose?. WWW-doucement. <http://blog.digitaltutors.com/unity-udk-cryengine-game-engine-choose/>. Referred 10.4. 2016.

Hiscott, Rebecca. 2014. 10 Programming Languages You Should Learn Right Now. WWW-doucement. <http://mashable.com/2014/01/21/learn-programming-languages/#gl6J3bZOskqL>. Updated 21.1.2014. Referred 16.4. 2016.

Geldonyetich. 2012. MMO Game Balance. WWW-doucement. <http://geldonsgaming.blogspot.fi/2012/07/the-impossible-goal-of-balanced-skyrim.html>. Updated 18.7.2012. Referred 16.4. 2016.

Unity Documentation, 2016. Unity Manual, Working in Unity, Creating Gameplay, Scenes. WWW-doucement. <http://docs.unity3d.com/Manual/CreatingScenes.html>. Referred 12.4. 2016.

Unity Documentation, 2016. Unity Manual, Working in Unity, Creating Gameplay, GameObjects. WWW-doucement. <http://docs.unity3d.com/Manual/GameObjects.html>. Referred 12.4. 2016.

Unity Documentation, 2016. Unity Manual, Working in Unity, Creating Gameplay, Transform. WWW-doucement. <http://docs.unity3d.com/Manual/Transforms.html>. Referred 12.4. 2016.

Unity Documentation, 2016. Unity Manual, Physics, Physics Overview, Colliders. WWW-document. <http://docs.unity3d.com/Manual/CollidersOverview.html>. Referred 12.4. 2016.

Unity Documentation, 2016. Unity Manual, Unity Graphics, Graphics Overview, Materials, Shaders & Textures. WWW-document. <http://docs.unity3d.com/Manual/Shaders.html>. Referred 12.4. 2016.

Unity Documentation, 2016. Unity Manual, Animation, Animation Reference, Animation Controller, Animation States. WWW-document. <http://docs.unity3d.com/Manual/class-State.html>. Referred 12.4. 2016.

Pearson, Charles. 2014. Learning NGUI for Unity. Birmingham: Packet Publishing.

Alex, Parker. 2013. Mobile Game UI. WWW-document. <http://www.riaxe.com/downloads/game-asset/>. Updated 21.10.2013. Referred 17.4. 2016.

Wenderlich, Ray. 2012. Introduction to AI Programming for Games. WWW-document. <https://www.raywenderlich.com/24824/introduction-to-ai-programming-for-games>. Updated 27.12.2012. Referred 17.4. 2016.