Catch The Kites
A Lightweight Android Game

Submitted By
Woaraka Been Mahbub
ID: 2012-2-60-033

Md. Tanzir Ahasion
ID: 2012-2-60-036

Supervised By
Md. Shamsujjoha
Senior Lecturer
Department of Computer Science & Engineering
East West University

A Project Submitted in Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
EAST WEST UNIVERSITY

April 2017

ABSTRACT

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 smart phones and tablets. The project we have developed is a game for Android platform. The name of the game is "Catch the kites". In this report we proposed the design and implementation of the game where our player will run in X axis and collect as much as kites for increasing scores and also player has to avoid enemies so that he can't lose points. We design and develop useful application with user interfaces by using; extending and creating own layouts and views.

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. So, the project has been developed in Unity 3D by using C# Programming Language.

DECLARATION

I hereby declare that, this project has been done by us under the supervision of **Md. Shamsujjoha, Senior Lecturer, Department of Computer Science and Engineering, East West University**. I also declare that neither this project nor any part of this project has been submitted elsewhere for award of any degree or diploma. Any material reproduced in this project has been properly acknowledged.

Supervised by:

_____

Md. Shamsujjoha
Senior Lecturer
Department of Computer Science and Engineering
East West University
Bangladesh

Submitted by:

_____

Woaraka Been Mahbub
ID: 2012-2-60-033
Department of Computer Science and Engineering
East West University

_____

Md. Tanzir Ahasion
ID: 2012-2-60-036
Department of Computer Science and Engineering
East West University

LETTER OF ACCEPTANCE

This Project entitled **"Android Game(Catch the Kites)"** submitted by Md. Tanzir Ahasion (ID:2012-2-60-033) and Woaraka Been Mahbub(2012-2-60-033) to the Department of Computer Science and Engineering, East West University, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on 13th April, 2017.

Supervisor

_____

Md. Shamsujjoha

Senior Lecturer
Department of Computer Science and Engineering,
East West University, Dhaka, Bangladesh

Chairperson

_____

Dr. Ahmed Wasif Reza

Chairperson (Acting) and Associate Professor
Department of Computer Science and Engineering,
East West University, Dhaka-1212, Bangladesh

ACKNOWLEDGEMENT

We would like to express our gratitude to our parents. Their diligent support and perpetual inspiration towards study since the early stage of our education, has placed us at the edge of our graduation degree. I believe that whatever we have achieved and whatever we are going to gain are owing to our parents.

We would like to pay homage to our supervisor Md. Shamsujjoha, His cordial directions have kept us on the right track from the very first day of supervision. Whenever, we came up with complicated issues, he guided us the simple way to resolve the issues. Besides, we are grateful to all our course directors for providing us with contemporary insights from the field of system development and implementation.

Our special thanks to all our friends, colleagues for their continuous inspiration and guidelines throughout my study period in East West University.

Moreover, we heartily thank to my family members for their financial supports for our study. Without their support, our study in this university could have been a dream, nothing more. We are profoundly grateful to our Creator that we have been in touch with and guided by such great individuals in the world.

TABLE OF CONTENTS

Page

Abstract……...i

Declaration……...ii

Letter of acceptance……iii

Acknowledgements……iv

Chapter

1. Introduction……01

1.1 Overview……...01

1.2 Motivation……01

1.3 Objectives……..02

1.4 Contribution……...02

2. Background Study……03

2.1 What is Android?.....03

2.2 Game Engines Today……...……..04

2.3 Programming Languages……...06

2.4 Game Basic Rules……..07

2.5 Game Sense……08

2.6 Game Object……..08

2.6.1 Transform……...08

2.6.2 Collider……..10

2.6.3 Materials, Shedders and Texture……..……...11

2.7 User Interface (UI)……12

2.8 Animation State Machine……..……13

2.9 Enemy (AI)……14

Chapter **Page**

3. Implementation……16

3.1 Overview……...16

3.2 Game Asset……..…….……17

3.2.1 Character Assets……...……..17

3.2.2 Ground Assets…….……...17

3.2.3 Kites Assets.....19

3.2.4 Enemy Assets…….19

3.2.4 Others Assets…….20

3.3 Starting with Ground……..21

3.4 Starting with Player..……..24

3.5 Camera Movement...……..29

3.6 Starting with Kites ……30

3.7 Enemy (AI)……32

3.8 Score……..34

3.8.1 Highest Score……...……..35

3.9 Moving Platform and Others…...……..36

3.10 Building and Mobile Button……..37

3.11 User Interface (UI)……...……..39

3.12 Building the Game……40

4. Final Result…..……40

5. Conclusion and Future Work……41

5.1 Conclusion…….42

5.2 Future Work…...…….42

References……43

Chapter 1

Introduction

1.1 Overview

In this project, we used the Unity game engine to create an endless 2D game which purpose is to catch kites. During this project, developers got familiar with most of the features of Unity engine and use them as much as possible in our process. Since the game was not only made with game engine, we 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, we will start with basic theoretical framework of the game design. Then we will introduce the Unity engine. After that, we will describe the implementation levels of the project. The implementation contains full details of the game process. Finally, we 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.

1.2 Motivation

The main motivation of this project is to explore the concepts of mobile application development in android. Android is the world's most popular operating system for mobile devices and tablets. It's make a revolution in the world of technology. Most of android applications are popular which are useful and entertaining for user. So we wanted to create a game which is developed by android using Unity game engine. If designed properly, a mobile version based this game could be more fun for user.

Thus, the following points describe the motivation for a mobile version game.

1.3 Objectives

The main objective is to develop a game which can be run any android platform device. For developing this game we will able to:

- Build our own Android game which can be run in other platform too like iOS, windows etc.
- Explain how the game is built.
- Explain all of our scripts for the game.
- Explain the software we use for making the game
- Explain the programming language(C#) and how it used in the game.
- Developed the game user friendly which will make the user entertain.

1.4 Contribution

To develop the game all the requirement were collected. Software, hardware and game framework requirements are also discussed in the chapter to initialize a gaming environment. After that we move into our project to discuss about it in details and also show the result.

- We have collected all the software from the internet and downloaded them.
- We learned about game engine and how to make game in android platform.
- We learned the programming language for making the game.
- We learned little bit about graphics design.
- We tested our game and it passed in all the method we applied.

For our game we installed Unity3D and Visual Studio 2015 for making our game. We also installed Photoshop 2017 for our graphics design. We learned all of these by the help of our honorable supervisor and internet. After making our game, we tested our game. When we found any problem, we solved that problem. We also tried our game in different mobile version and we found that it worked all mobile devices.

Chapter 2

Background Study

2.1 What is Android?

Android is a mobile operating system developed by an American company Google. The name "Android" comes from the term android, which is robot designed to look and act like a human. It is used by several smart phones and tablets. This means you can easily look for the information on the web, watch videos, search for directions, write emails etc on your phone, just as you would do on your computer [1].

The Android operating system (OS) is based on the Linux kernel. Unlike Apple's iOS, Android is open source, meaning developers can modify and customize the OS for each phone. Therefore, different Android-based phones often have different graphical user interfaces GUIs even though they use the same OS.

Android phones typically come with several built-in applications and also support third-party programs. Developers can create programs for Android using the free Android software developer kit (SDK). Android programs are written in Java and run through a Java virtual machine JVM that is optimized for mobile devices. There are lots of versions on Android. Users can download and install Android apps and games from Google Play and other locations.

The latest estimates say there are 12 million mobile app developers worldwide, representing more than half of the total global developer community, and almost half focus their attention on Google Android. The total number of developers is expected to reach 14 million by 2020.

The report also looks at which platforms developers are concentrating on. Android comes out on top, with 5.9 million developers considering Google’s platform as their number one choice, and 2.8 million concentrating on Apple’s iOS. It’s noted most developers write for multiple platforms.

In Asia Pacific, 2.2 million mobile developers focus on Google’s Android operating system first, with only 500,000 addressing Apple’s iOS before anything else. It’s a

reversal in North America, where iOS gets attention first, and the gap between the two is more than 200,000.

I am showing a graph in Figure 2.1 which will tell us that how fast the android apps are increasing day by day [2].

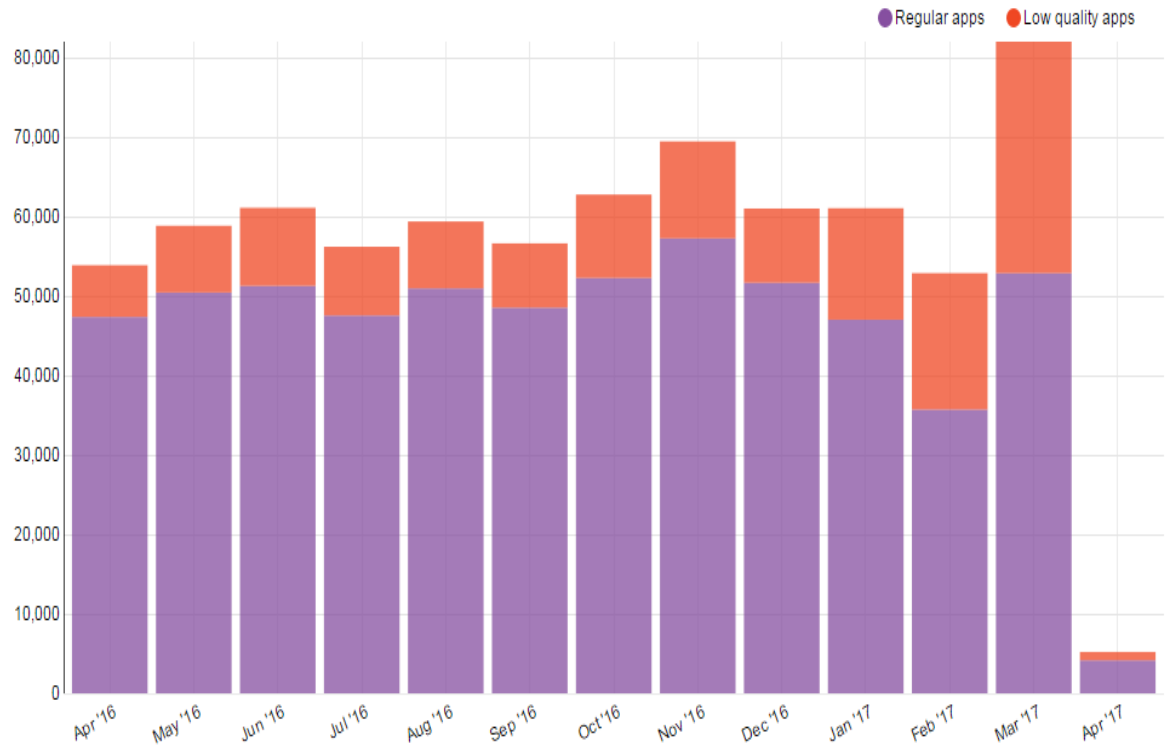

Figure 2.1: New android apps per month

2.2 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.

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.

In the current industry, the game engines for the mainstream of the game companies are Unity3D, Source 2, Unreal Engine and Cry-Engine [3]. Starting with Source engine, source engine is relatively backward compare with other engines. The graphic technology has fallen behind, but the source engines 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.

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

Unity3D 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.

For our game project we used Unity3D to build our game because it is the most affordable game engine for game developers. As it is the most popular game engine, so we learned this game engine easily. Because lot of tutorials, feathers, Colum's, forums etc are found in the internet.

2.3 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.

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.

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.

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.

For our game project we used C# programming language to build our game because C# is a simple and object oriented programming language which derived from C and C++. Unity3D also supports C# language for making games. Visual studio 2015 has a unity tool to link up with Unity3D and C#. The person who knows C and C++ languages, he can easily catch this C# language. For those reasons use this language in our game project.

2.4 Game Basic Rules

In our game, there is a player whose aim is to catch kites. The player is running in the top of the buildings and there is an empty space between two buildings. Player can go one building to another by jumping. Also there are some moving platforms which take the player one direction to another. If the player fall in the empty space the game become over. Before the game being over, player can catch kites which are flying and falling. By catching each kites player get some points. Player main aim to go left side to right side to catch kites but player can also move right side to left side if player feels that there are kites in the left side which the player left. But in this game, our camera will follow our player movement. If the player move right side our main screen also move right side. But the screen doesn't move left side if the player move left side. Also our screen will automatically run from left side to right side if the player doesn't run and screen speed will be increasing, it is depending on player position. If the screen is running faster than player and the screen pass the player then game will also be over. Player will get some scores for surviving in the game. Of course player will get many points if it catches kites. The final score will be the combine of catching kites and how much times player survived. We also add some enemies which are birds. These birds make problem for the player to run properly. This game is an endless game. The game will not be over until the player fall down or pass the screen.

2.5 Game Sense

Before starting for 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 [4].

2.6 Game Object

In the process of making the game, the Game-Object is one of the most important objects that we will use in Unity. The definition of Game-Object is a little bit vague. It could be any object in the game that the player can interact with. Before Game-Object becomes a specific thing, we need to give the object some special properties, so that the Game-Object 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 Game-Object [5]. Moreover, the Game-Object typically contains some basic properties, and each property will have different effects on the Game-Object. Game-Object can also affect the significance of the object in the game. Therefore, understanding each property is very important.

2.6.1 Transform

The Transform functions of Game-Object in Unity are to control its physical parameters. Transform is used to control Game-Object's position in the game. The Rotate formation is used to control Game-Object's orientation. The Scale is used to control Game-Object'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 2.2

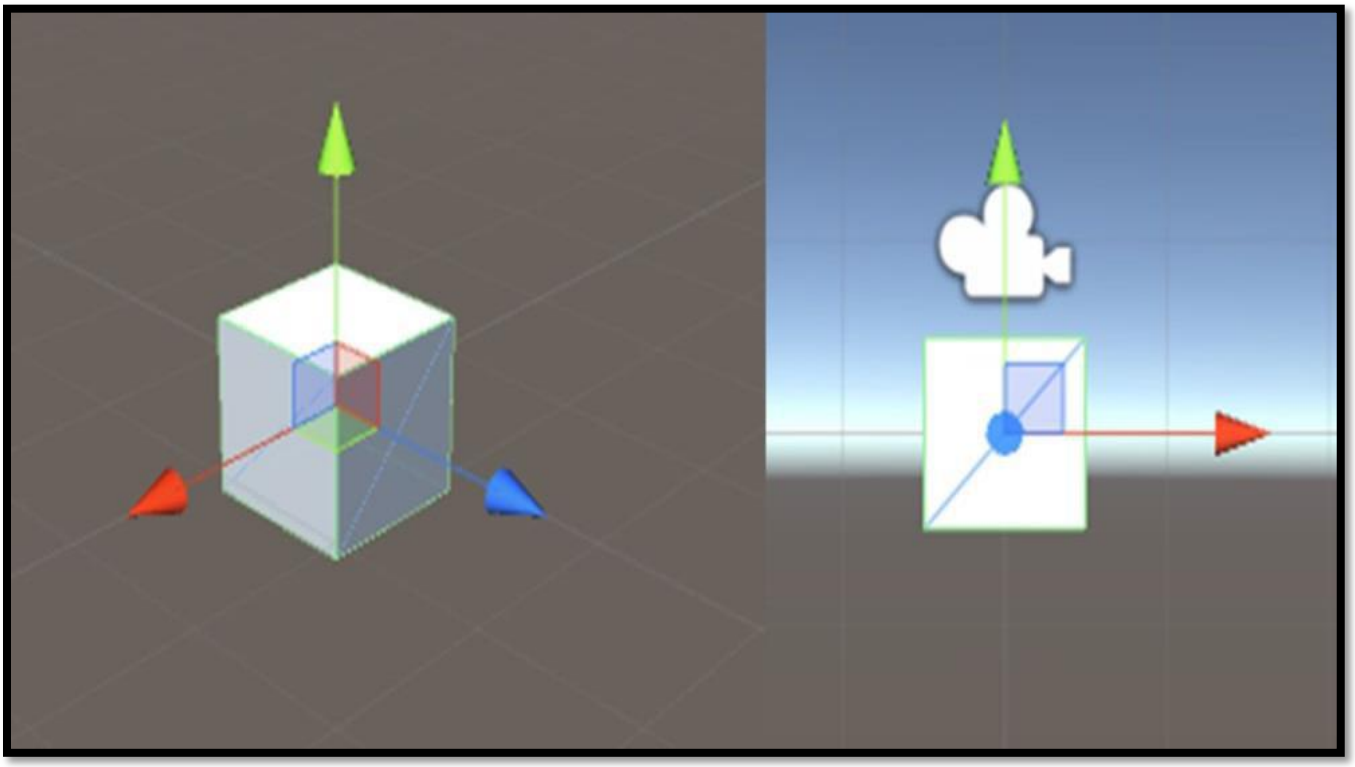

Figure 2.2: X, Y and Z axes in 3D and 2D space

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 Game-Object in C#, the basic form within a script is shown in figure 2.3

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

Figure 2.3: The basic script form of the Game-Object Transform

In Figure 3, the position and scale are stored as vector3 because Game-Object 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.

2.6.1.1 Transform.parent

In Unity, parenting is one of the most important concepts during the game production process. For example, if Game-Object is a parent of another Game-Object, its Child Game-Object will make the same changes of its transform like its Parent does [6]. In game, transform.parent can be used for binding the Game-Objects or camera with the character or other Game-Objects. The transform.parent is basically script as Figure 2.4 shows.

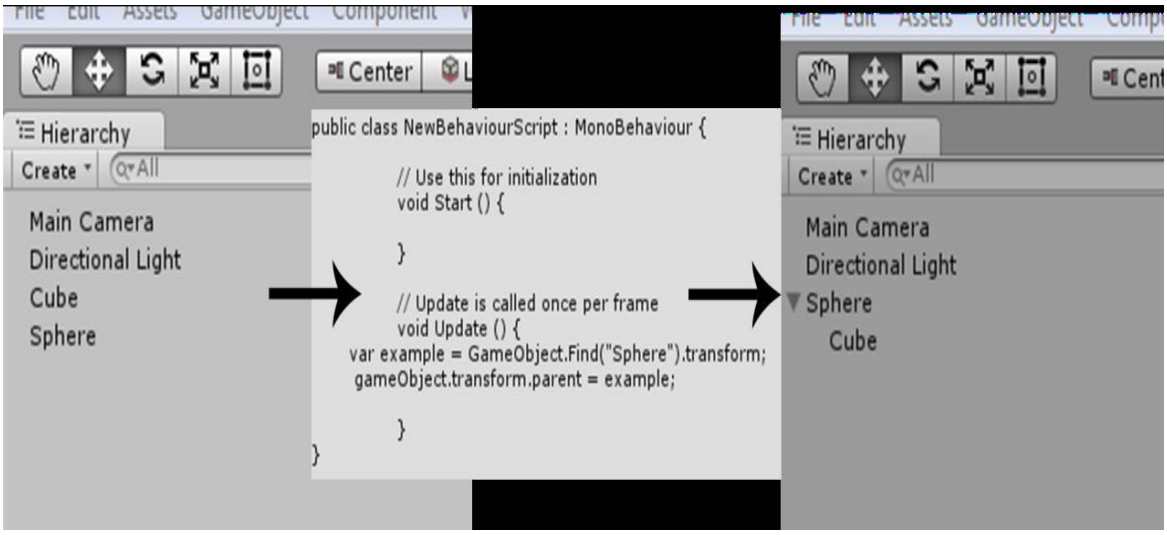

Figure 2.4: Work process of the Transform.parent

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

2.6.2 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 Game-Object. 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 are shown in Figure 2.5

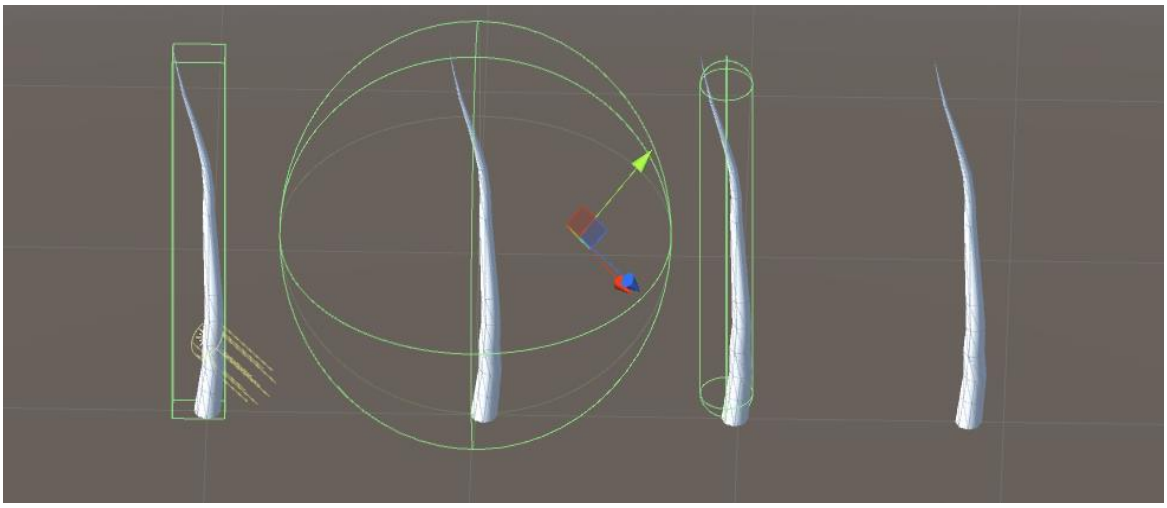

Figure 2.5: Capsule Collider, Mesh Collider, Wheel Collider and Terrain Collider

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 Game-Objects. Moreover, when Box Collider and the Sphere Collider are not enough to fit the Game-Object, we can use Mesh Collider to match the mesh of the Game-Object. 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 Game-Objects 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 [7].

2.6.3 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 Game-Object. In most cases, we use the Standard Shader to the Game-Objects. 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.

2.7 User Interface (UI)

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, free, and simple 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 2.6 shows.

Figure 2.6: Example of Game UI

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 plug-in for Unity3D. In many games, game designers prefer 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 Game-Objects 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.

2.8 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. Also, interaction between the game characters and the game objects should be adjusted in the Animation 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 2.7 shows.

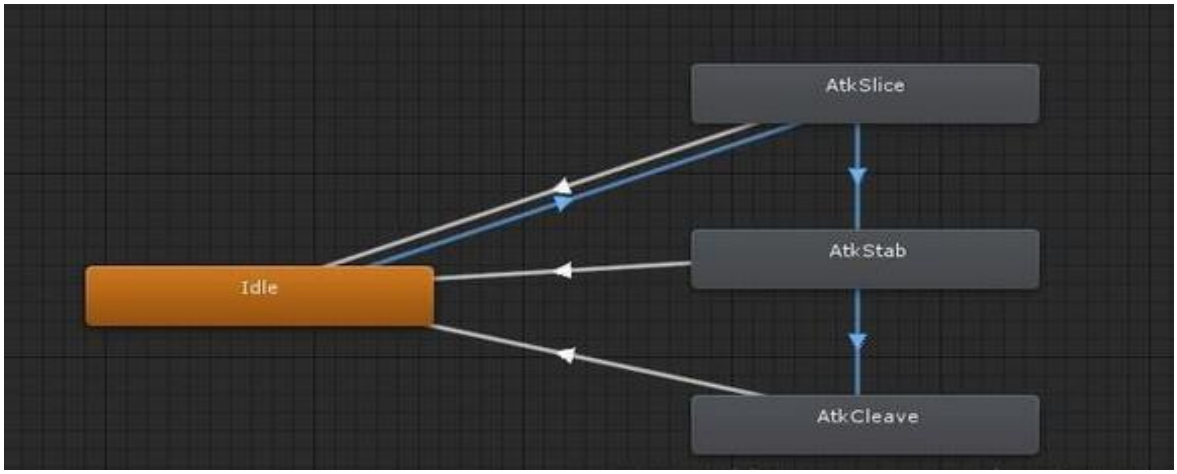

Figure 2.7: Animation state machine in Unity3D

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, running, jumping, landing, sliding, attacking, being injured and so on. All of these actions are called as states. We need to connect 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 running animation state and keeping the game character in running 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.

2.9 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.

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. 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, and 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 game-play 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 combinations 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 2.8 shows, for example.

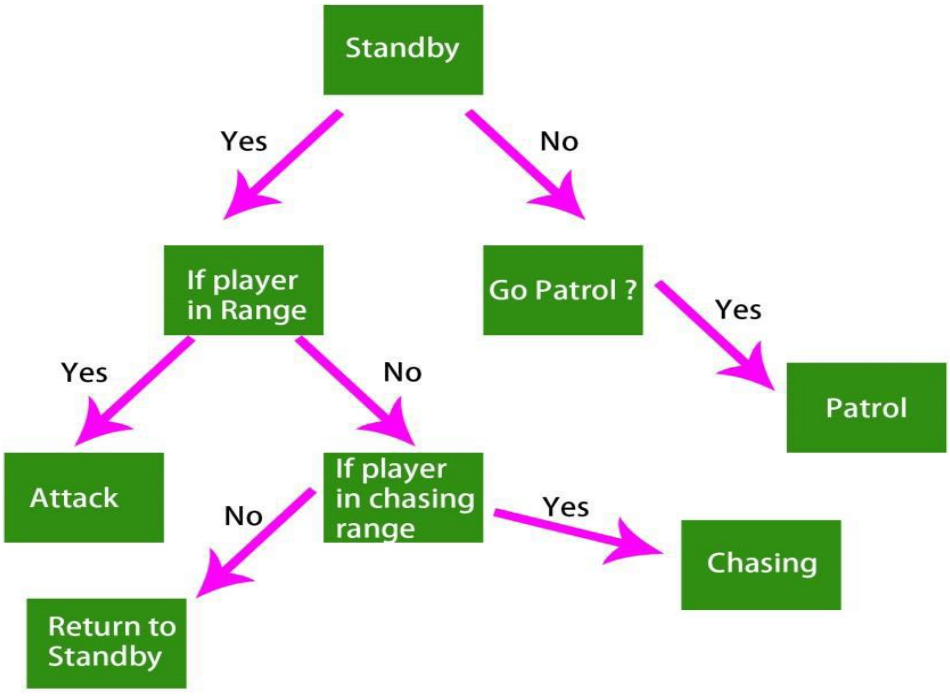

Figure 2.8: Example of AI logic diagram

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.

Chapter 3

Implementation

3.1 Overview

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, we need to create an endless running game whose aim is to catch kites 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 mainly 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 over. Our screen also moves from left to right. If the screen passes the player then the game will over. Also there are some birds which will disturb the player movement. We are showing an example flowchart in the Figure 3.1 which will tell our scoring method.

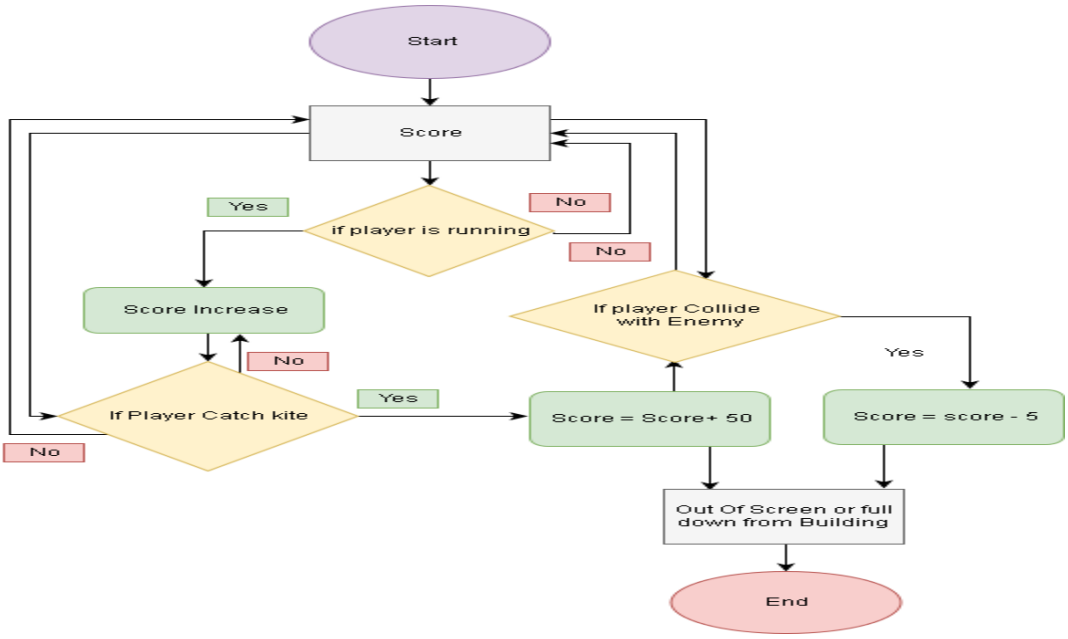

Figure 3.1: Flowchart (Score)

3.2 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 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 game-play 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, enemy etc.) are taken from internet and edited them by Photoshop during the project.

3.2.1 Character Assets

For the character, we made multiple sets of images for the character's action such as idling, jumping, running and sliding. 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, we need to adjust the timeline of each image in the animation state machine to make the action become natural. In Figure 3.2, 3.3, 3.4, 3.5 we are showing our character assets which action is idling, running, jumping and sliding.

Figure 3.2: Character assets (action idle)

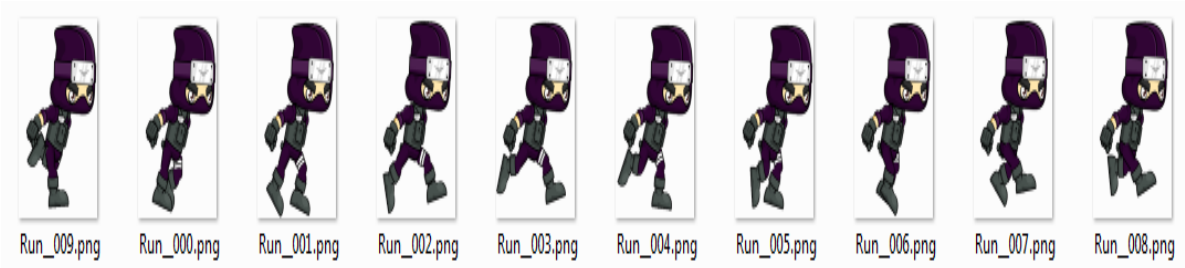

Figure 3.3: Character assets (action running)

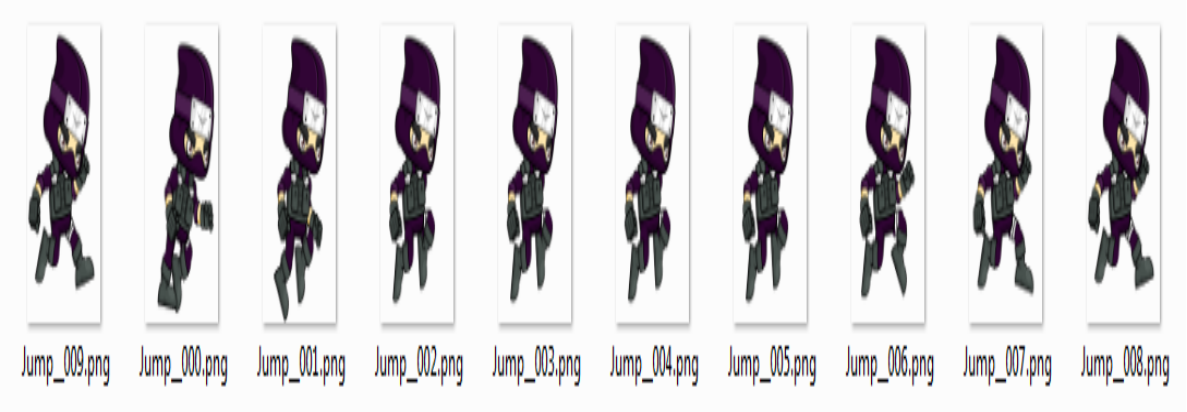

Figure 3.4: Character assets (action jumping)

Figure 3.5: Character assets (action sliding)

3.2.2 Ground Assets

In this game our ground are some buildings. Our player will run from the top of these buildings. The ground is randomly generated 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. In figure 3.6 we are showing some of our grounds.

Figure 3.6: Ground assets

3.2.3 Kites Assets

In our game kites are mainly used as coins. We made multiple sets of images for the kites'. The animation will be produced by looping the image set from the first to the last image. If the player can catch a kite it will get some points. In our game, kites will generate automatically. These kites will fall from the sky and it will fall from left to right screen. We used different types and color kites so that our game looked nice. In the figure 3.7 we are showing some kites assets.

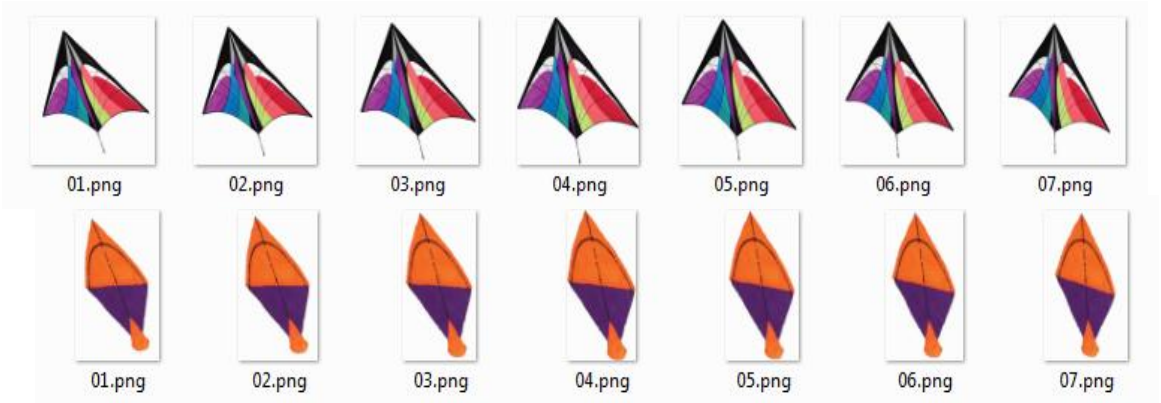

Figure 3.7: kites assets

3.2.4 Enemy Assets

In our game we use some enemies whose are some birds. These birds animation also will be produced by the same logic as we described previous. These birds disturb our player movement. If the player collides with these birds, player will lose some scores. In Figure 3.8 we are showing our enemy asset.

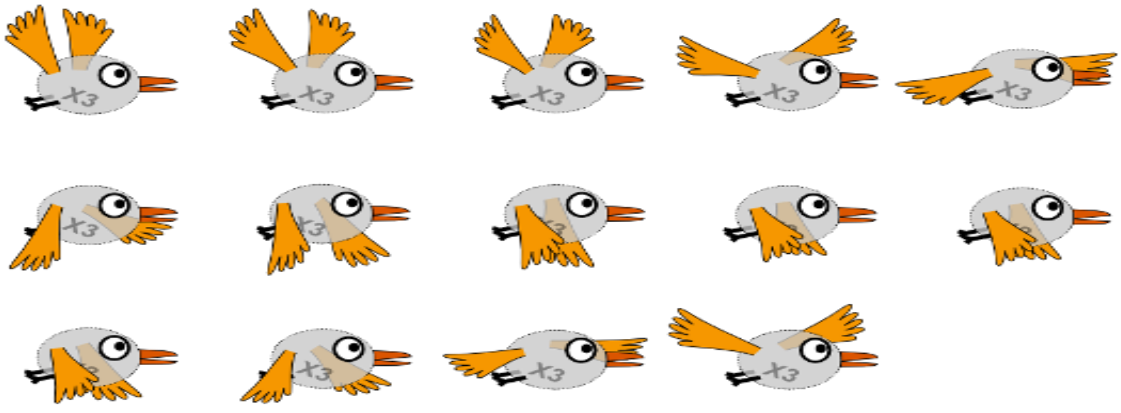

Figure 3.8: Enemy asset

3.2.5 Other Assets

In our game we also use some other assets. We have used a background asset. We also use some moving platform assets whose job are taking the player one side to another. We also used helping assets which job is help the player for jumping in the high buildings. In figure 3.9 we are showing our background asset and in figure 3.10 we show our moving and helping assets.

Figure 3.9: Background asset

Figure 3.10: Helping assets and moving assets

3.3 Starting with the Ground

Starting the game stage, we first need to create the basic ground to let the main character walk on it. Our character will run on the top of buildings. First, we 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 polygon collider 2D to the ground, so that the ground can contact with the main character. Also, the polygon collider 2D 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. Figure 3.11 shows the ground setting.

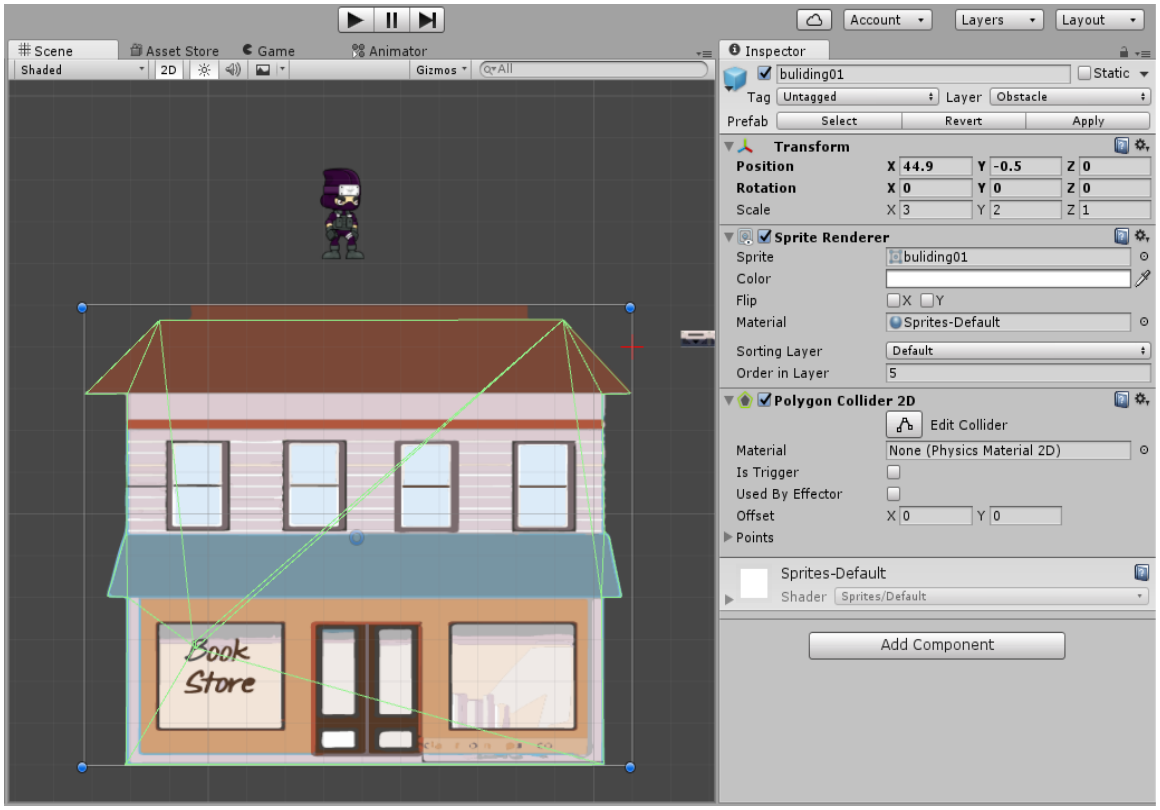

Figure 3.11: The basic setting of the Building prefab

Also, we need to create the prefabs of the buildings 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, we have created some prefabs called Building and moving platform. From these

building prefabs we created some labels which will generate buildings automatically. In Figure 3.12 and 3.13, we will show the auto label creation.

Figure 3.12: Building auto creator (before running project)

Figure 3.13: Building auto creator (after running project)

For building auto creator we have written a C# script called Spawning_infinite_tiles under the game object. With the help of this script, auto buildings are generated. This auto creation is depending on the player position. For the memory optimization, we will delete the previous buildings which the player passed. In this script, we also give a condition for not creating the previous building continuously. In Code 3.1 and 3.2 we are showing our script.

```
void Start ()
{
    activetiles = new List<GameObject>();
    playerTransfrom = GameObject.FindGameObjectWithTag("Player").transform;

    for(int i=0; i<tileOnScene; i++)
    {
        SpawnTile();
    }
}

// Update is called once per frame
void Update ()
{
    if(playerTransfrom.position.x - safeZone > (spawX - tileOnScene * tileLength))
    {
        SpawnTile();
        DeleteTile();
    }
}
```

Code 3.1: Auto building creation (1)

```
private void SpawnTile (int prefabIndex = -1)
{
    GameObject go;
    go = Instantiate(tilePrefabs[RendomPreFabIndex()]) as GameObject;
    go.transform.SetParent(transform);
    go.transform.position =new Vector3(1 * spawX,spawY,0);
    spawX += tileLength;
    activetiles.Add(go);
}

private void DeleteTile()
{
    Destroy ( activetiles[0] ); //destroy the list tiles
    activetiles.RemoveAt(0);
}

private int RendomPreFabIndex()
{
    if (tilePrefabs.Length <= 1)
        return 0;

    int randomIndex = lastPerfab;
    while (randomIndex == lastPerfab)
    {
        randomIndex = Random.Range(0, tilePrefabs.Length);
    }
    lastPerfab = randomIndex;
    return randomIndex;

}
```

Code 3.2: Auto building creation (2)

When we running the game initially it start with start ( ) function. In this function there is a for loop and this loop will be running until the last value of tileOnScene which value we initialized. In the loop there is a function called SpawnTile ( ). This function we create a Game-Object variable. In this variable we will call our RendomPreFabIndex ( ) function and create clone in the variable randomly. In the RendomPreFabIndex ( ) function we put a condition that if our total building is one then we do not need to randomize. There is a while loop in the RendomPreFabIndex ( ) function condition is that if last prefab and new prefab are same then it will generate randomize otherwise the function will return clone to the Game-Object variable. In the DeleteTile ( ) function, the tile will be deleted which the player passed. Then every time the Update ( ) function will be called.

3.4 Starting with the Player

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 ten 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 3.14

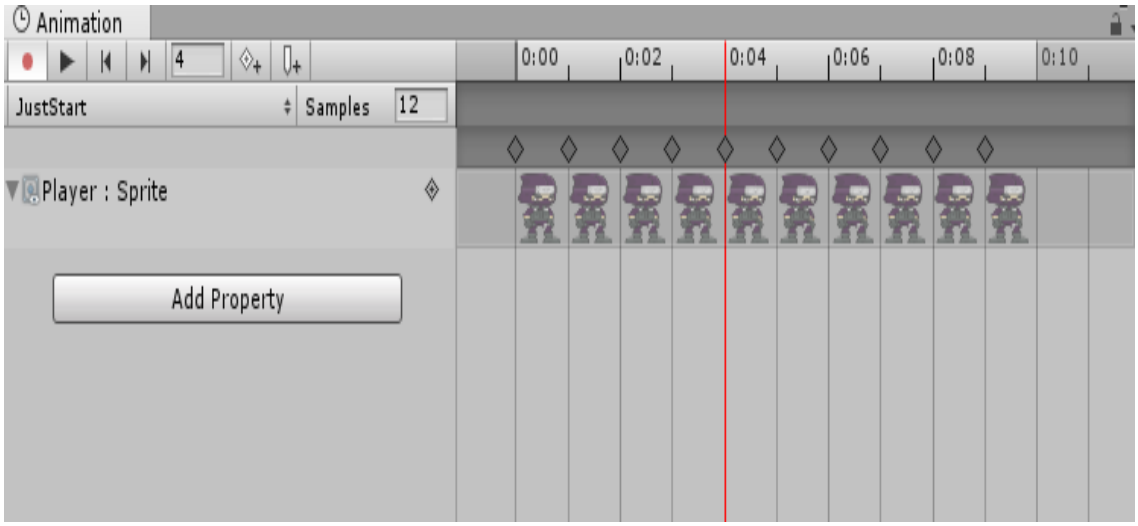

Figure 3.14: The animation of running in frames

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. 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 3.15

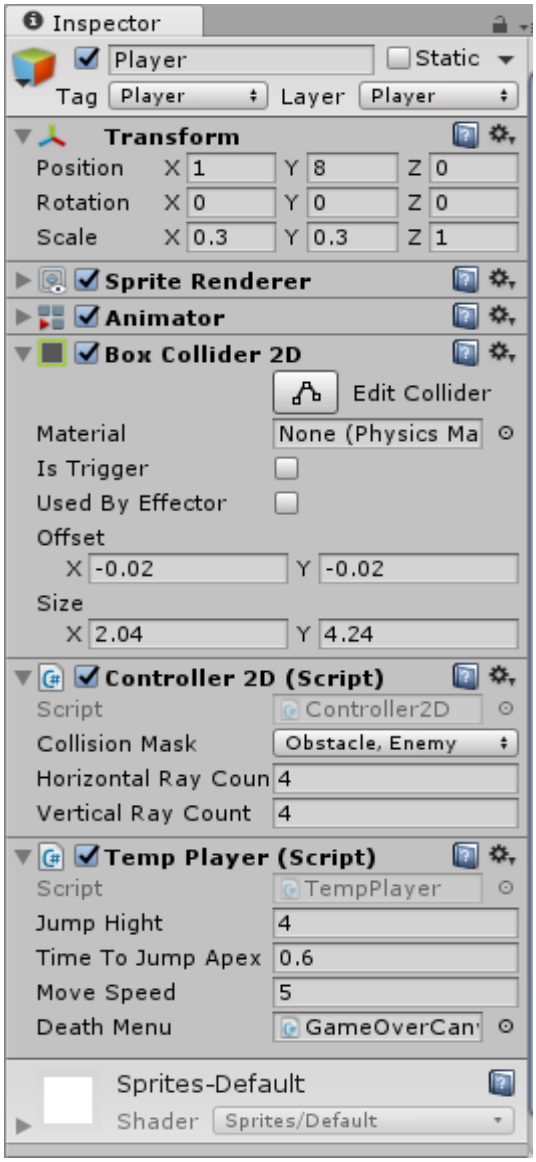

Figure 3.15: The properties of the player

After the animation and properties of the character is done, we need to add control to the character with some scripts. In the player we used two C# script. The basic control in this

game is written in the TempPlayer script. The basic control is when we pressed the Space button our player will jump over the gap to the other grounds and when we pressed Left Control button then our player slide. Also, in this game the character can only jump once at a time. In Code 3.3 we are showing our control script of the player.

```
private void HandleInput()
{
    if (Input.GetKeyDown(KeyCode.Space))
    {
        myAnimator.SetTrigger("Jump");
    }
    if (Input.GetKeyDown(KeyCode.LeftControl))
    {
        myAnimator.SetTrigger("Sliding");
    }
}
```

Code 3.3: The basic control

As the user is controlling the movement of our player so we also give some control in our game. For moving left to right we will use right arrow and left to right we will use left arrow. In Code 3.4 the movement control is given.

```
{
    horizontal = Input.GetAxisRaw("Horizontal");
    input.x = horizontal;
}
```

Code 3.4: The basic moving control

Since the character will run, jump and slide during the game process, the animation needs to switch between those actions. Therefore, we create two layers. One is base layer and another is jump layer. In the base layer we are showing our slide movement and make a transition from running to sliding and sliding to running. We also create a sliding. Figure 3.16 will show the base layer.

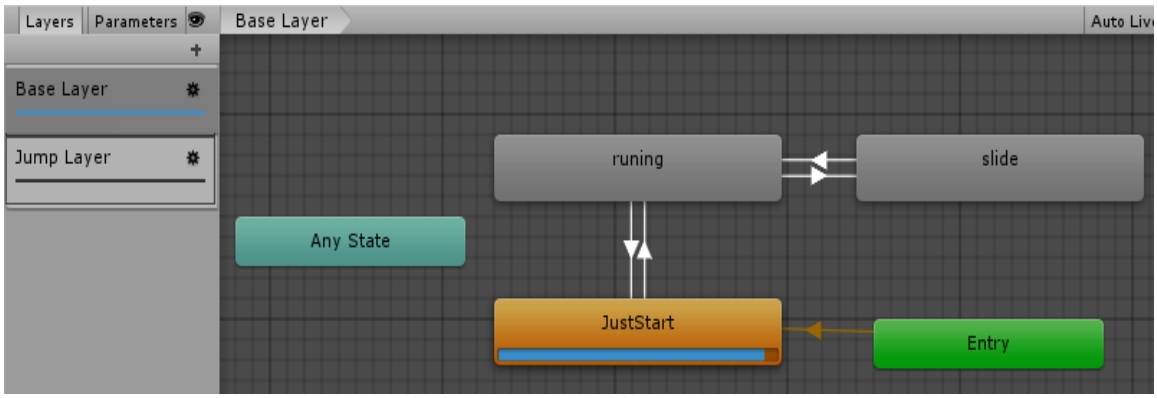

Figure 3.16: Base Layer

In the jump layer, we are showing our jumping and landing animation. The running and jumping animations need to connect to each other in the Animator by the transition components. Afterwards, all the transitions need to be added a parameter in the condition setting called jump, so when the character takes an action and reaches the condition, the animation to another one. In the same way, we added landing so that our player can land after jumping and its parameter is landing. Figure 3.17 is showing our Jump Layer.

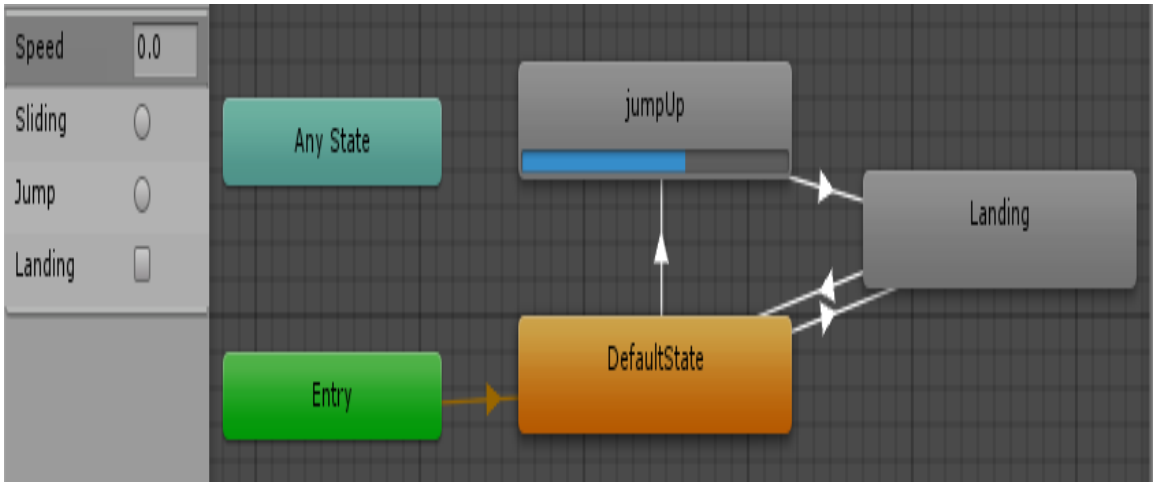

Figure 3.17: Jump Layer

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 code is given in Code 3.5.

```
private void HandleLayeout()
{
    if (!controller.collisions.below)
    {
        myAnimator.SetLayerWeight(1, 1);
    }
    else
    {
        myAnimator.SetLayerWeight(1, 0);
    }
}
```

Code 3.5: The Layer Code

In this game our player will be died if the player fell in the empty space between two buildings and if our screen passed our player. In the figure 3.18 we are showing the dead since.

Figure 3.18: The sense of death

For the dead sense, we need to write script so the player will die if it fell down or pass the screen. In the Code 3.6, the death code is given. In the code first we are checking our Y axis means the player fell or not. Then we check X axis means our camera is passing our player or not. If any condition becomes true then our game will be over.

```
private void Death()
{
    if (transform.position.y < -8 || transform.position.x < (targetCamera.position.x - 15))
    {
        IsDead = true;
       // PlayerPrefs.SetInt("HighScore", playerTransform);
        deathMenu.ToggleEndMenu();
    }
}
```

Code 3.6: Game over code

3.5 Camera Movement

The main job of our camera is to follow our player movement. When our player is moving in the right side then our camera will also move in the right side. We also give some speed for our camera so that it is moving all time and camera speed will be increased. If our screen passed our player then the game will be over. For those work we write a C# script called CameraFlow. The script is showing in the code 3.7

```
void LateUpdate()
{


    if (PauseMenu.Instance.IsPaused==true)
    {
        return;
    }

   float valueGot = calculateSpeed();

        if (target.position.y < -7.5 || target.position.x < (transform.position.x - 15))
        {
            transform.position = new Vector3(transform.position.x, transform.position.y, transform.position.z);
        }
        else
        {
            if (target.position.x > transform.position.x)
            {
                transform.position = new Vector3(target.position.x, Mathf.Clamp(target.position.y, yMin, yMax), transform.position.z);
                camreaPosX = target.position.x + valueGot;
            }
            else
            {
                transform.position = new Vector3(camreaPosX, Mathf.Clamp(target.position.y, yMin, yMax), transform.position.z);
                camreaPosX = transform.position.x + valueGot;
            }

         }
}

private float calculateSpeed()
{
    float result;

    result = (MoveSpeed + (transform.position.x/100)) / value ;
    return result;
}
```

Code 3.7: Camera Script

In LateUpdate ( ) function first we are checking that our camera is passed our player or not. If passed the player then our camera will freeze that means game will over. Otherwise, our camera will be moved and camera speed will be increased. CalculateSpeed ( ) is the function for increasing our camera speed based on player position.

3.6 Starting with the Kites

In our game kites are mainly used as coins. If I catch more kites my score will be increased. In the kites we used animation. We also add some component like Animator, Script, Rigidbody 2D and Polygon Collider 2D. These are shown in the Figure 3.19

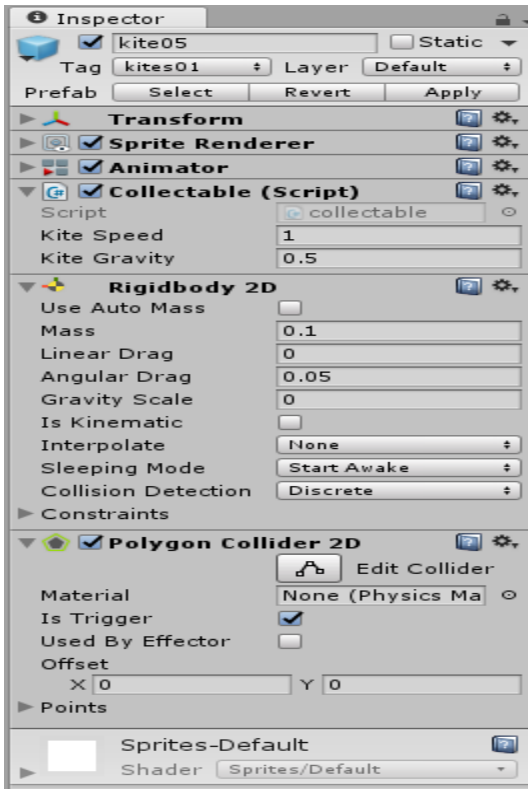

Figure 3.19: Basic setting of Kites

In the figure, the Transform component is used to set the position of the kite. The Animator component is used for the animation of the kite. Rigidbody 2D component is used for all the physics work like mass, gravity, angular frequency etc. The Polygon Collider 2D component is a Collider for use with 2D physics. The Collider's shape is defined by a freeform edge made of line segments, so you can adjust it to fit the shape of the Sprite graphic with great precision. In the Polygon Collider 2D we used Is Trigger so that it can interact with the player. In the kite we used a C# script called kite-Script. With the help of this script we are collide the kites with my player, add score after colliding and then destroy the kites. In code 3.8, we are showing our kite's script.

```
private ScoreScript kiteManager;
private int kitePoints = 50;
private float myPlayer;

private Rigidbody2D myKite;

[SerializeField]
private float kiteSpeed;
[SerializeField]
private float kiteGravity;


void Start () {
    kiteManager = GameObject.FindObjectOfType<ScoreScript>();
    myKite= GetComponent<Rigidbody2D>();
    //      kiteManager = GameObject.Find("ScoreScript");

}

// Update is called once per frame
void FixedUpdate ()
{
    myKite.velocity = new Vector2(1 * kiteSpeed, 1 * -kiteGravity);
}

private void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Player")
    {
        kiteManager.KiteCatch(kitePoints);
        Destroy(this.gameObject);
    }
}
```

Code 3.8: Kite's Script

In the script, we initialize our kite point 50 that means if we catch a kite we get 50 point. We also add a kite speed in X axis which tells how moving speed of the kite from left to right. We also add a gravity speed which indicates that how it falling in the land. OnTriggerEnter2D is called when another object enters a trigger collider attached to this object. Collider2D other is used for the other Collider2D involved in this collision. Here we check if the player collides with kite it will get 50 point and then destroy the kite. We will talk about our score later.

3.7 Enemy (AI)

In our game, our enemy is birds as we talked earlier. It disturbs our player movement and if player collides with birds, it will lose some points. For these birds, we make animation of birds' movement so that it looks realistic. These birds will be created automatically like our buildings were created. We also add some other components in the bird which is shown in the figure 3.20

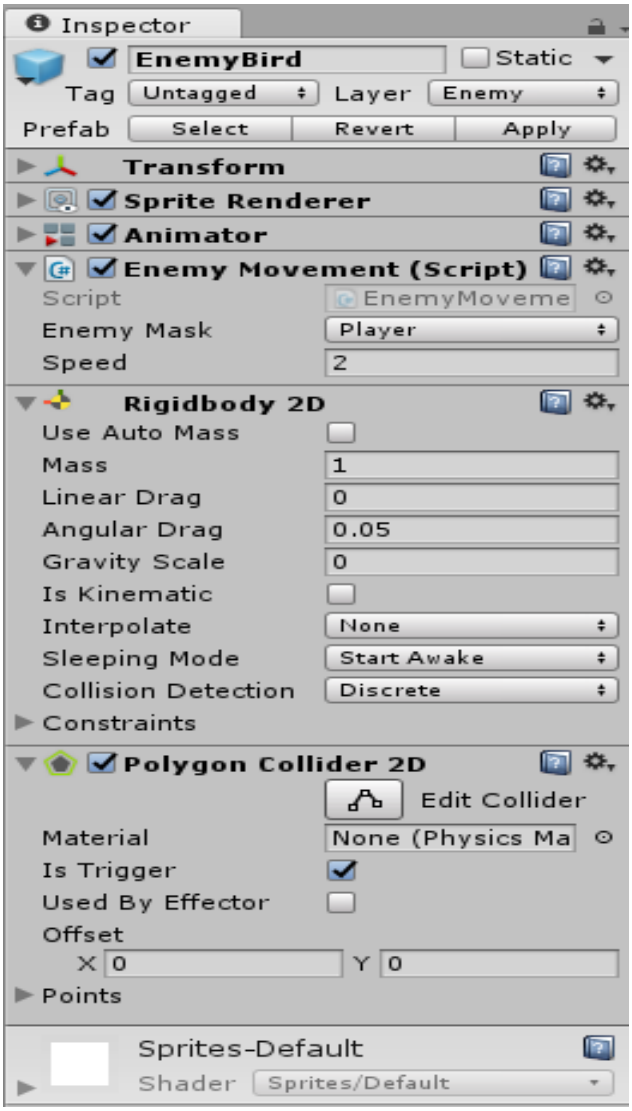

Figure 3.20: Basic setting of Enemy

The transform component is used to set the position of the bird so that we can set our enemy in the actual position. Animator component is used for animation. Rigidbody 2D component is used for all the physics work like mass, gravity, angular frequency etc. The Polygon Collider 2D component is a Collider for use with 2D physics. In the

Polygon Collider 2D we open Is Trigger so that it can interact with the player. For the movement, we used a script called EnemyMovement which is shown in code 3.9

```
void FixedUpdate()
{
    Vector2 enemyVelocity = enemyRigidbody.velocity;
    enemyVelocity.x = enemyTransform.right.x * speed;
    enemyRigidbody.velocity = enemyVelocity;

    if((FacingRight && transform.position.x > (myFlipPoint+ Random.Range(4 , 12)) )
        || (!FacingRight && (transform.position.x < myFlipPoint - Random.Range(5, 12))))
    {
        FlipPlayer();
    }
}

private void FlipPlayer()
{
        FacingRight = !FacingRight;
        Vector3 theScale = transform.eulerAngles;
        theScale.y += 180;
        transform.eulerAngles = theScale;
}

private void OnTriggerEnter2D(Collider2D other)
{
    if (other.tag == "Player")
    {
        kiteManager.DecendingScore(emenyPoint);
      //  Destroy(this.gameObject);
    }
}
```

Code 3.9: Enemy Script

In the script, our bird is initially direction is right side. In the FixedUpdate ( ) function there is an if condition which compare bird's position and bird's direction. With the base of the condition next time the bird change his movement. FlipPlayer ( ) function is used to change the bird's facing. And OnTriggerEnter2D (Collider 2D other) function is used to check whether the player collide with bird or not. If collide the player will lose points.

3.8 Score

In this game, our score is used for catching kites and for surviving in the game. In our game we used total score and the kites count. We kept kites count because in future we'll add some feather like achievements. For the score we used some UI in the game like image and text. For the score we used a C# script called Score-Script. In Code 3.10, we are showing our score script.

```
public void IncrimentScore(int points)
{
    score = points + mykitespoint - enemyPoints;

    scoreText.text = score.ToString();
    EndScoreText.text = score.ToString();
}

public void KiteCatch(int kitesPoints)
{
    mykitespoint += kitesPoints;
    totalCatchKites++;
    kiteCatchText.text = totalCatchKites.ToString();
    EndKiteCatchText.text = totalCatchKites.ToString();
}

public void DecendingScore(int points)
{
    enemyPoints += points;
}

private void highScoreUpdate()
{
```

Code 3.10: Score Script

In the IncrementScore (int point) function, our total score will be adding the position of our player, adding kites point and deduct enemy point. In the Kiteatch (int kitesPoints) function, we count our total kites those the player caught. In figure 3.21 we show our score UI.

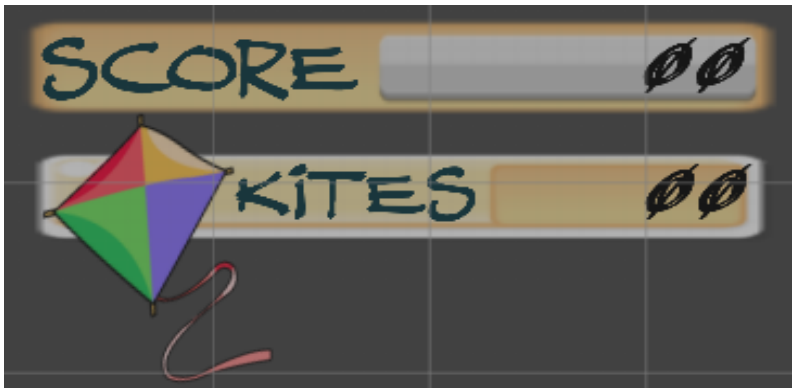

Figure 3.21: Score UI

3.8.1 Highest Score

In our game we use highest score UI which tell us the total highest score and total kites the player catch in a game. For that we used a function in the score script that is shown in the Code 3.11

```
private void highScoreUpdate()
{

    if (PlayerPrefs.GetInt("HighScore") < score)
    {
        PlayerPrefs.SetInt("HighScore", score);
    }
    if (PlayerPrefs.GetInt("HighKites") < totalCatchKites)
    {
        PlayerPrefs.SetInt("HighKites", totalCatchKites);
    }

    //  showing HighScore in HighScore DashBoard ..........//
    HighScore.text = PlayerPrefs.GetInt("HighScore").ToString();
    HighestKite.text = PlayerPrefs.GetInt("HighKites").ToString();
}
```

Code 3.11: Highest Score Script

In the function highScoreUpdae ( ), we are checking whether current score and current kites are greater than the previous score. If true then we put our current score into high score and current kites into high kites.

We also create a score UI and Highest score UI that is shown after the game being over. These UI is shown in the figure 3.22

Figure 3.22: Game over UI

3.9 Moving Platforms and Others

In our game, moving platforms is used to help our player from one side to another. In these moving platforms we used some component which is given in the figure 3.23

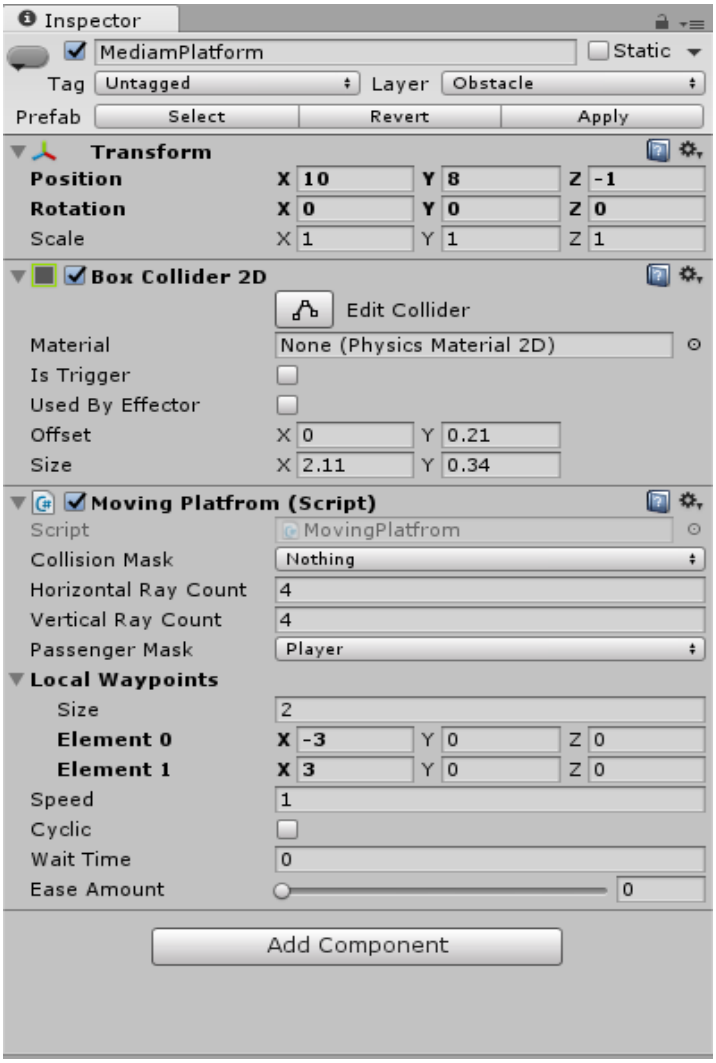

Figure 3.23: Basic setting of Moving platform

In the Moving platform script we are using passenger mask which tell us that who will pass with the moving platform. In the local Waypoint we give the size 2 that means our moving platform will move in two sides. Also, speed is available in Local Waypoint for speed of our moving platform. If we click on cyclic then moving platform move cyclic. Also, there is wait time which indicate how much time it wait in one side.

We have also used some helping platform which job is to help the player for jumping in the high building. For the helping platform we just use Box Collider 2D so that our player can land on them.

3.10 Building the Mobile Button

As our game is implemented for the android, so we need to add buttons for control the game like movement, jumping etc. For that we are using some buttons. We made these buttons by the help of Photoshop. The buttons are shown in the Figure 3.24

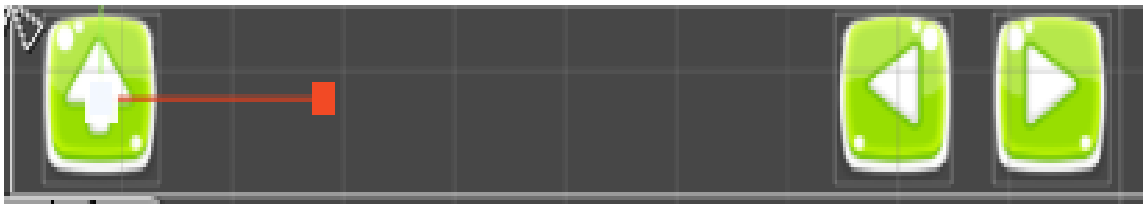

Figure 3.24: Button for Player Movement

The right side buttons are used for control our movement and the left side button is use for jumping. There are some components we have to use in the buttons so that they work. Figure 3.25 is the basic setting of these buttons. In the Event Trigger script we made some function for jumping, moving etc. We call those functions in the Event Trigger. Then we'll get all the function under player. Then we select our function. In the figure we choose jumpBtn for jumping.

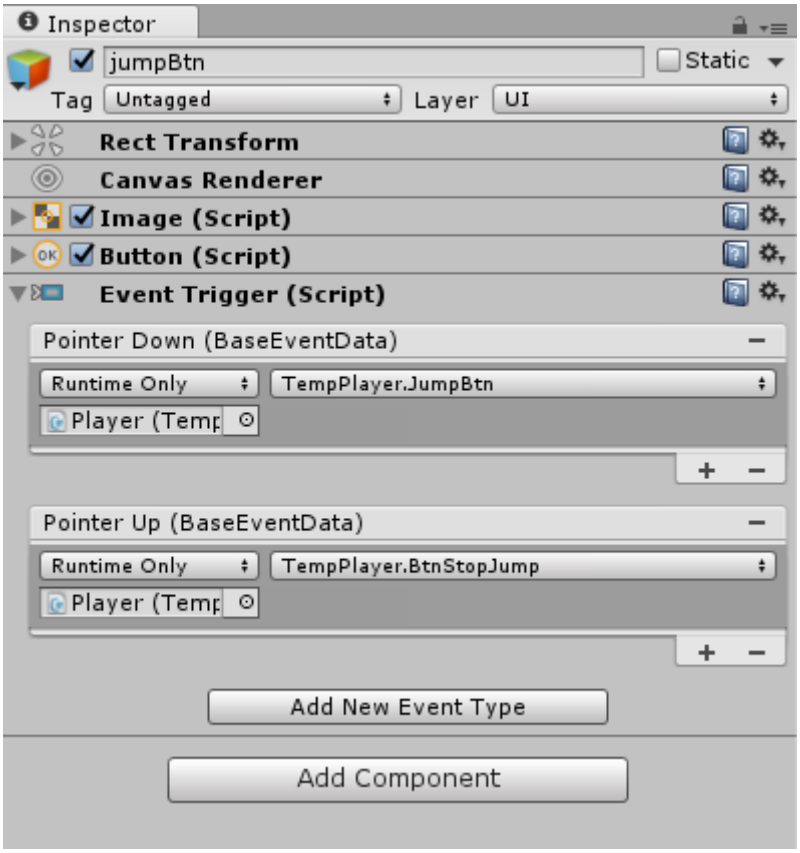

Figure 3.25: Basic setting of Button (jump)

For the button we wrote some C# functions which are shown in the Code 3.12. In this code we have JumpBtn and RunBtn. When we'll press those buttons our then in these functions the jumpbtn and this.move become true. When we left these buttons then BtnStopRun and BtnStopJump will be called and they will stop these buttons work.

```
public void JumpBtn()
{
    myAnimator.SetTrigger("Jump");
    jumpbtn = true;
}

public void RunBtn(float direction)
{
    this.direction = direction;
    this.move = true;
}
public void BtnStopRun()
{
    this.direction = 0;
    this.btnHorizantal = 0;
    this.move = false;
}
public void BtnStopJump()
{
    this.jumpbtn = false;
}
```

Code 3.12: Code for Mobile Button

3.11 User Interface (UI)

We have also created some UI for the game. We created a main menu which is shown in the figure 3.26. In the figure we can see that there is a catch button which job is to play the game. Score button show us the highest score and the highest kites we caught. The help menu is used to help the user how to play the game. Also we have some music on and off button, about button, share on social media button and quit button. Also we used some animation on it.

Figure 3.26: Main Menu

We also made a pause menu which is shown in the figure 3.27. In this menu we kept resume, restart, main and quit menu. We also kept music on/off button in there.

Figure 3.27: Paused Menu

3.12 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 3.28. Since Unity 3D is a cross-platform engine, Unity 3D supports almost every game platform in the game market.

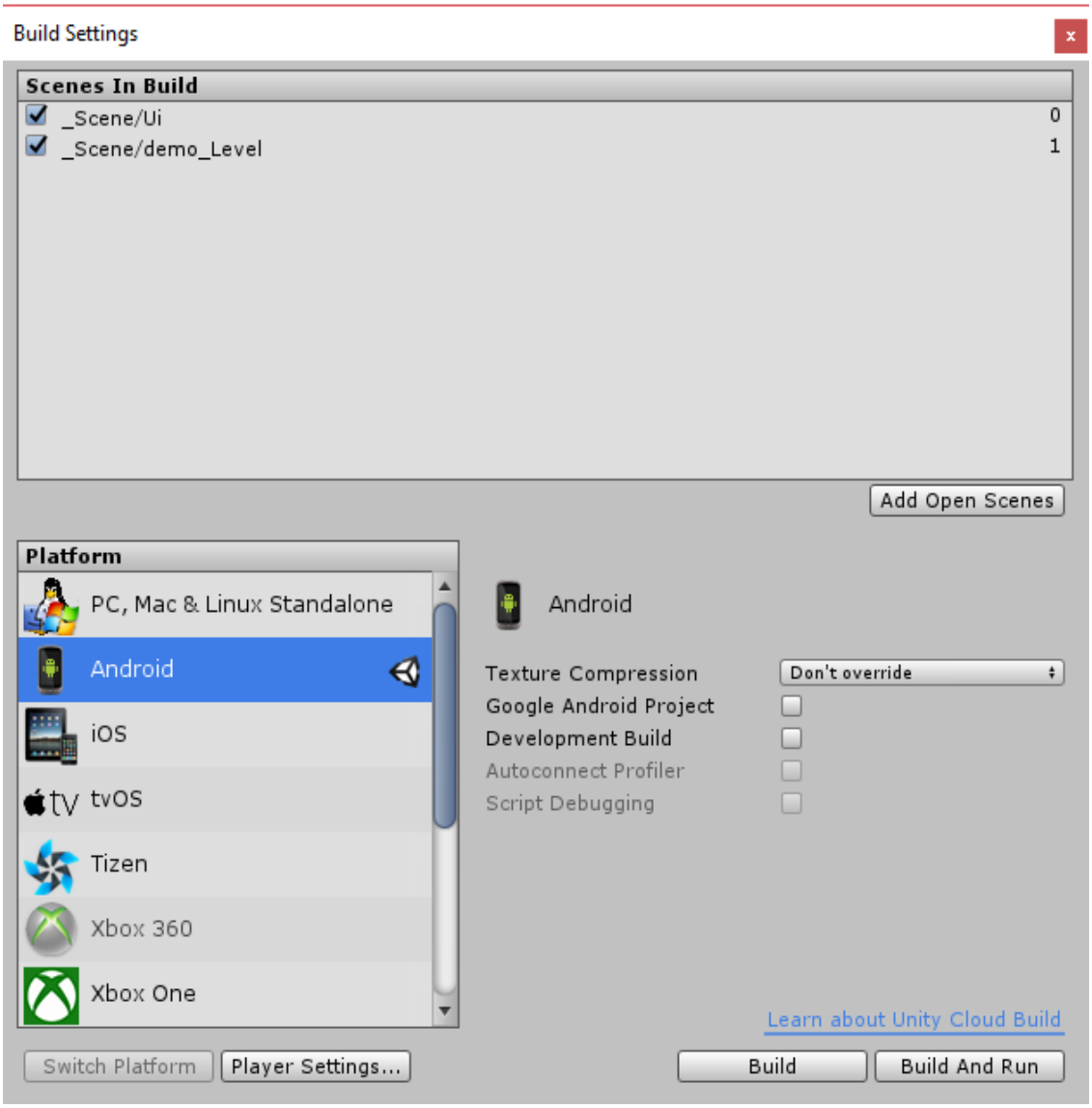

Figure 3.28: Build and Setting tool in Unity 3D

Figure 36 shows the game scenes in the Scenes in Build window which includes the game start menu and the main game scene. Also, we choose to run this game on the Android platform, since another platform might need some advanced configurations. When building is finished then we need to copy the APK file to our Android phone and install it. Then play the game.

Chapter 4

Final Result

During this project, all the basic features were achieved completely endless running game which aim is to catch kites on an Android platform. All the game assets were created and modify by ourselves with Photoshop and Blender 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, movement by our created controller buttons on screen. The game points are increasing based on player position of X axis. Also, there are kites those have to catch by the player for getting bonus points 50.Also if the player collides with the birds which are our enemy then the player will lose some points 5. So here our main goal is to catch kites as much as possible and also survive in the game avoiding the birds. The player dies, if he fall down from the building and also if the screen passes the player. 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 APK file which is created from the Build & Setting tool.

Chapter 5

Conclusion and Future work

5.1 Conclusion

The aim of the study was to create a 2D endless running game which aim is to catch kites with the Unity 3D game engine. During this project, we 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 game-play scene from this scene.

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, we have improved our knowledge and skill of working with the Unity 3D game engine. Also, we consolidate our knowledge of programming with the C# language.

5.2 Future Work

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 we'll add some achievement for the user so that the users get interest for the game. Also we'll add other labels on the game. We'll try to make our design more realistic. Next time we'll try to convert our game in 3D.

References

1. Matthew Bolton, What is Android? A beginner's guide, http://www.techradar.com/news/phone-and-communications/mobile-phones/what-is-android-a-beginner-s-guide-975482 (05-03-2017)

2. Mathijs Vogelzang, Uwe Maurer, Number of Android applications, https://www.appbrain.com/stats/number-of-android-apps (12-03-2017)

3. Wikimedia Foundation, List of game engines, https://www.en.wikipedia.org/wiki/List_of_game_engines (20-03-2017)

4. Unity Technologies, Scenes, https://docs.unity3d.com/Manual/CreatingScenes.html (23-03-2017)

5. Unity Technologies, Game Object, https://docs.unity3d.com/Manual/GameObjects.html (25-03-2017)

6. Unity Technologies, Transform, https://docs.unity3d.com/Manual/class-Transform.html (27-03-2017)

7. Unity Technologies, Collider, https://docs.unity3d.com/Manual/CollidersOverview.html (29-03-2017)