LAPIN AMK
Lapland University of Applied Sciences

CREATION OF GAME

Unity game for PC platform

Helavirta Antti
Xie Yuting

Bachelor’s Thesis
School of Business and Culture
Business Information Technology
Bachelor of Business Administration

2017

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

School of Business and Culture
Business Information Technology
Bachelor of Business Administration

Authors	Antti Helavirta & Yuting Xie	Year	2017
Supervisor	Johanna Vuokila		
Commissioned by	-		
Title of Thesis	Creation of game		
Number of pages	49 + 7		

Currently, the gaming industry is divided into two different development processes, i.e. Indie development and traditional corporate development model. Even though not completely different from each other, major distinctive differences between these development models are present during the multiple development processes. A majority of traditional development companies tend to only focus on financial benefits, where Indie developers are focusing on passion in gaming and industry in itself.

There are several objectives in this thesis work. Firstly, differences in traditional development and Indie development are examined from commercial, financial and development point of view. During this research work, a game demo was also created to have more in depth view towards Indie development. Further, multiple different development diaries and guidelines were examined, based on information gained through these guidelines the demo game was created. After the development process, the demo game was tested according to industry testing standards. Moreover, new programming language of C-Sharp was learned, in order to have more in depth views to the most common game development language. Additional help from developers’ friends and contact personnel were used during testing phase. In order to conduct industry standard testing procedures, people with different experiences towards gaming had to be used during the testing phase.

Constructive research method was used during the research phase of this thesis work. However, during the development phase, spiral model was used in order to have successful project. Majority of developed scripts created during this project were unnecessary to be presented during this development work and, therefore, they were left out from the text format.

The end result of this development work is a playable demo version of a game designed by the developers However, the game will not be presented to the public, due to further development in the future.

Key words Indie, C-Sharp, Development, Scripting, Unity

CONTENTS

SYMBOLLS AND ABBREVIATIONS

RPG	Role Playing Game
JS	JavaScript
C#	C-Sharp
ESA	Entertainment Software Association
NPC	Non-Playable Character
GUI	Graphical User Interface

FIGURES, PICTURES AND TABLES

FOREWORD

Gameplusjames guidelines provided by him through YouTube platform played a huge part in completing this thesis work. Without his guidelines, the starting phase of this thesis development work would have taken considerably more time. Test subjects contributing to black box testing and grey box testing deserve a special thank you for participating in this stage. The information gained through this process contributed to a huge amount of information necessary for this project.

Unity sub Reddit community deserves a thank you for helping to complete this project and helping on the solutions created for this project. The community enabled a fast paced process creating layers and objects within the game environment. Special thanks go to “Nick X” for providing solutions for the beginning of the animation development; without her contribution this process would have never been successful. Special recognition is given to “Feihti” for providing resources throughout this process.

1 INTRODUCTION

This chapter discusses the background of this thesis topic, together with the motivation, scope, methodology and limitations. The sources and structure of this thesis will also be discussed in this chapter.

1.1 Background and Motivation

The current indie game industry is blooming due to a highly popular names being released constantly by individual developers, “Binding of Isaac”, “Don’t starve”, “Risk of Rain”, “Faster Than Light” and multiple other names are trending in gaming industry. Before this thesis project began, both developers were already passionate gamers and Indie game enthusiasts. Currently Indie game industry has released names on almost every possible game category. However, one sector is lacking names, that is role playing games (hereinafter RPG). Shared passion between both developers towards RPG peaked interest towards this project of developing games, and the choice made towards designing and creating an RPG meant tremendous amount of research towards the subject of game development.

Possibilities in Indie game development industry are limitless. However, since one field is missing mentionable names in the industry, the developers of this work found it appropriate to develop a game specifically to this category. The project in itself had a huge impact from an Indie developer “Gamesplusjames”. This developer created a similar type of game planned for this project and posted guidelines to YouTube for other developers to learn from his project. (Gamesplusjames 2015a.)

1.2 Scope and Objectives

This thesis work is focusing on the development process during Indie games and Unity programming. Since Unity utilizes two different programming languages JavaScript (hereinafter JS) and C-Sharp (hereinafter C#), the programming part of this thesis will be focusing on the C# language due to past experience with JS and available sources of game development using C#. Due to the nature of programming using C#, there are multiple different possible ways of achieving the desired goal. However, different programming solutions will not be provided for this work. This thesis development work will work as an experimental work for the developers, rather than as a guidebook for new developers. Due to the high amount of existing guides and online courses, the developers found that making a guidebook would be unnecessary. The testing phase of this thesis work will be conducted by the thesis work developers and few selected close friends of the developers. In other words, this testing phase was concluded in a closed circle to prevent unnecessary distribution of the development work.

The research part of this development work is to conduct research in differences between traditional and Indie game development processes. Additionally, research towards case development work was conducted, focusing mainly on PC platform due to hardware requirements unmet by the tools available during this project.

Graphical design of this thesis work is a secondary objective, and during this development work graphical aspects are considered after the desired functions have been met. Due the lack of skills in graphical designing, during this process a close friend of the developers aided with the graphical design part, and free online sources were used to get graphical objects.

Furthermore, the testing phase will be conducted to demo version created for this thesis development work. During the testing phase, possible flaws and bugs scripted during the development phase are found out. Due to the scale of this work, only a prototype version was developed. However, the game has most of the functionalities designed and developed but will not be implemented into the prototype version. Further development towards this prototype version will be conducted after this thesis project.

1.3 Research Methodology and Limitations

During this thesis research project, multiple research methods were discussed as possibilities to reach the objectives set for this work, among them the constructive research method suggested by Kasanen (1993) and improved by Lukka (2000). The constructive research method suggested by Kasanen and Lukka is divided in six steps as seen from figure 1. However, for this work, steps 1,2,3,4 and 6 were found to be most suitable for this development work.

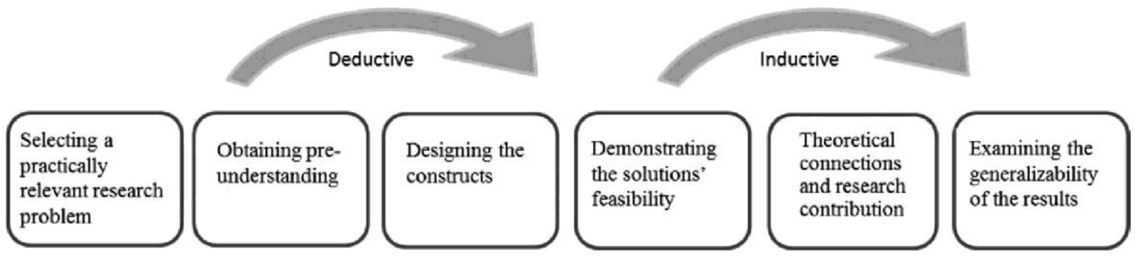

Deductive
Inductive
Selecting a practically relevant research problem
Obtaining pre-understanding
Designing the constructs
Demonstrating the solutions' feasibility
Theoretical connections and research contribution
Examining the generalizability of the results

Figure 1. Constructive Research Method (Kasanen & Lukka 2000)

The first and second step of this research method is discussed in design part of this development work. However, due to nature of this work, step two of this method is considered more important that step one. Step three is discussed in chapter four in this thesis work. Although, majority of the research results of step three are highly dependable on step two results, changes according to developers' personal view were made. The development part of this thesis work focuses on step four, Implementation of designed features is conducted in this part. Finally, step six of the research method is implemented in the testing phase of this development work.

In order to reach the objectives set for this development work, development diaries, Unity guidelines and documentation developed by Indie developers were analyzed in order to gain the necessary information for this work. Questionnaires, interviews and public polls were not necessary for this development work. However, during the testing phase outside help had to be used in order to gain the desired results.

The authors’ lack of experience in Unity development impacted the quality and complexity of features implemented in this work, although previous knowledge of gaming in general impacted on the final result of the development work. Additionally, previous of knowledge of other programming languages decreased the time required in order to learn C# language. Although, majority of the knowledge of C# language was gained through direct development work rather than studying from literature sources, guidelines and best practices provided by other developers were considered.

1.4 Research Questions

Three research questions are presented. Discussions of the research questions are provided.

1. What are the benefits of Indie development compared to Traditional development?

To answer this question, Indie game development point of view is analyzed from multiple different development diaries provided by developers. In order to accomplish this analysis, research towards Traditional development is done from sources provided by respected companies. Furthermore, to emphasize Indie development point of view, game demo was developed using guidelines and processes provided by other indie developers.

2. How chosen development tools effect the development process?

Current development tools used when developing games highly effect the development result. The research work thoroughly investigates differences between most popular development tools used in gaming industry. Furthermore, analysis of different programming languages used by development tools is also analyzed from same perspective as software.

1.5 Regarding Sources

The Programming phase of this work depends highly on the quality of the sources. Due to the nature of C# language, there is countless amount of books and material available today. However, online support of Unity and C# programming with Unity has much more to offer online compared to literature sources, since programming techniques and software are changing at a fast pace and, therefore, online sources for this project were more desired. Since Unity is open source software, the company itself is providing online guides, scripts, graphical objects and solutions free of charge. Therefore, Unity homepages will work as the main source of programming sources. Furthermore, other sources, including Reddit, YouTube and individual development diaries, were used to gain information about Unity functions and other possible solutions regarding the programming part of this thesis.

Furthermore, majority of the sources used in this development work are taken from individual development diaries, YouTube videos and Reddit community guides. Therefore, only online nicknames are available to be used when referring to the authors.

1.6 Structure of Thesis

The tools and software chosen for this project are discussed in chapter 2 of this thesis development work, decision concerning these choices will be concluded in this chapter. Chapter 3 is used to discuss differences in traditional development and indie game development, advantages and disadvantages of different development processes are emphasized in this part. Chapter 4 of the thesis discusses features planned and designed to be used in the case development work, research concerning these choices and decisions will be described in this chapter. Chapter 5 will discuss the implementation of the designed and planned features. This chapter will work as the main body of this thesis development work. Chapter 5 discusses about bugs and troubles encountered during the development phase, together with the solutions to these problems. Chapter 6 focuses on the testing part of this work, this includes discussion on the testing tools and methods of testing. Chapter 7 of this thesis draws conclusions concerning this development work.

.

2. TOOLS AND SOFTWARE USED

This chapter discusses the tools and software used when developing and scripting features designed and implemented in this work. Main focus of this part is on Unity platform due the nature of this development work.

2.1 Development Software

This project revolved highly around Unity development platform. However, other software had to be used in order to have successful development during the project. These software included Visual Studio, Tiled2Unity, Notepad++, Audacity and Photoshop. Unity as itself provides almost every function provided by other softwares that were used in this project. However, Visual Studio was used during the troubleshooting phase to gain much more elaborate solutions to programming flaws inside the scripts developed for this project. Visual studio is also the default software for Unity to respond to incase of programming errors. (Unity technologies 2016a.)

Tiled2Unity is a software used to create textures inside the game. This software is just used to convert already existing objects and textures, to a form where unity recognizes these as different layers and objects. Uploading files inside Tiled2Unity converts these into .tmx files, these files are recognized as texture files by Unity. (Barton 2016.)

Notepad++ was used as the primary script development tool. Due the lightness of the software, Notepad++ was favored over Visual Studio due to high processing power required by the Visual Studio. Notepad++ does provide troubleshooting in same manner as Visual Studio, however due the chosen programming language of C# which is not understood by Notepad++. (Ho 2016.)

Audacity software was used during the sound design and implementation phase, because Unity has its own sound mixing features, Audacity_was mainly used creating delay or shortening sound effects. Audacity software was not considered as a necessity for this project. However, due to previous knowledge of this software, it was used to ease the burden of sound development. (The Audacity team 2016.)

Photoshop had a minor role in this project. Due to online libraries providing objects capable to be used in this project, Photoshop was only used in case changes into objects were desired.

2.2 Platform and Unity Features

A large part of this development work was done inside Unity. However, features provided by Unity can overcome the need of scripting, therefore ease the load of work needed to have desired functions inside the development work. Every script written using either Notepad++ or Visual Studio was compiled using Unity to work in harmony with other scripts.

Unity recognizes two programming languages as a default, i.e. C# and JS. During this project, the focus was on C# language. Even though these languages provide the same functionality compared to each other, due to multi-paradigm nature of C# language it fits as the more suitable option for this project.

Unity functionality provides functions to facilitate the development of during the scripting phase. These functions include layering of different objects in the game, controlling setup for player movable objects, and mapping controls and object definitions. These functions play an important part of the project, since taking these aspects of development work out of the scripting phase degreases the chance of flaws in these aspects.

Layering different objects inside the game allows developers to assign objects to work as desired, from ground level to aesthetic objects inside the game. Layering requires to be defined in specific format .tmx. Tiled2Unity is software designed to change different objects to this format.

Player movable objects control settings defined by scripting, leaves high chance of implemented flaws done by the developer. Unity offers simplistic functions to be assigned to player movable objects, including weight, speed, durability, actions and directional movement. Directional and actionable controls can simply be defined to any desired controls. However, when defining these controls animation will be defined to have visual effects during actions. As seen from Figure 2, animation can be attached to functions in simplistic manner.

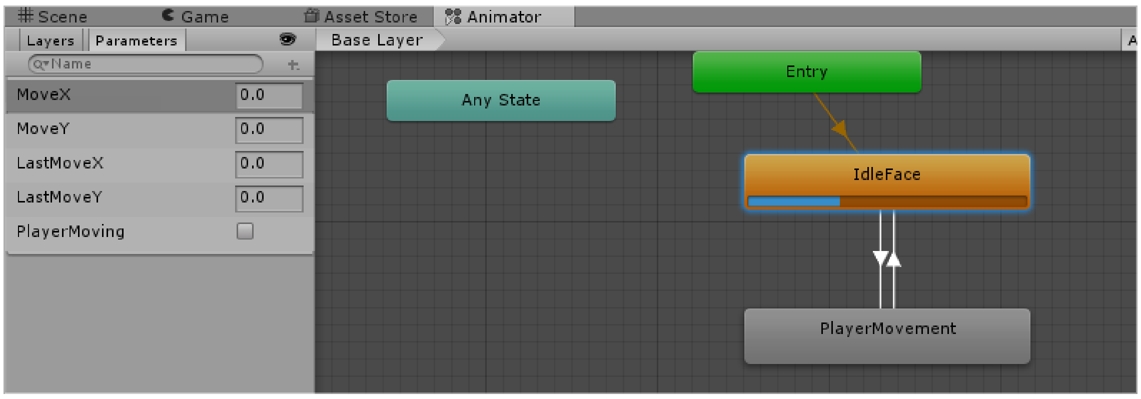


Figure 2. Animation Setup for Character Movement and Actions

Object definition is most simplistic, yet one of the most crucial parts of every development work. Without defining player movable character, enemies or any other object within the game, the objects cannot behave in desired matter. To determine immovable and indestructible objects to create boundaries within the game can have crucial part during development work, without defining boundaries of the game objects would mean game being literally unplayable.

2.3 Followed Protocols and Models

During the development phase, important factors for game development must be followed. Version control, development models and script consistency are the major factors for game development. These development factors are not unique for game development rather than common along all software development phases. Although C# is a multiparadigm language and allows differences in scripting models, when every script follows the same model it is more developer friendly when further developing the scripts. For consistency in this development work, guidelines for C# development provided by Mr. Lance Hunt fit this work perfectly. Even though his development standards are not necessary for game development, the standards provided by him work on this development work. (Hunt 2007.)

Version controlling when using Unity development platform is done easy for developers. Unity does not provide version system control unit by themselves but supports Perforce and Plastic SCM natively. Even though these software functionalities are not immediately in use when setting up first development project, Unity itself suggest using version control system and provides these for free through asset store. Since this project has multiple developers and everyone has the different programming style, the software tracks and records all changes made into the scripts by different users. This is done to ease the following of change in every script developed by the developers; when mistakes occur, these are easier to be reverted and changed back into the original form. (Unity technologies 2017b.)

When determining the models used when developing games, it is important to factor the designed game style. Since this development process is focusing on 2D RPG game, the spiral development model was chosen for this project. Even though this model is not only used in game development, as presented by Mr. Boehm, the spiral model is constant development and testing towards developed functions. (Boehm et al. 2014.)

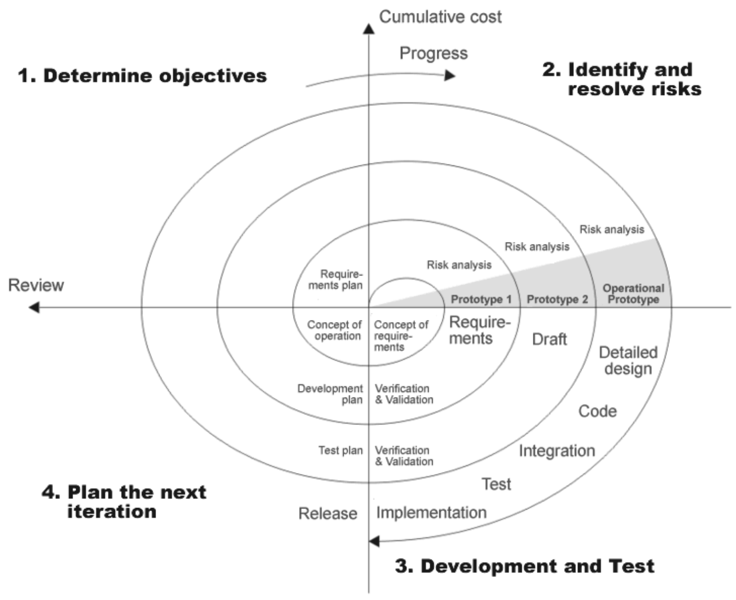


Figure 3. Spiral Model (Boehm 1988)

As seen from Figure 3, the model is quite similar when comparing to the waterfall model. The model works perfectly in to game and software development. When implemented correctly, development and testing of different parts of work is constantly conducted in the project.

3. TRADITIONAL DEVELOPMENT AND INDIE DEVELOPMENT

In this chapter, the difference between indie development and traditional development is discussed. Advantages and disadvantages of Indie development are heavily emphasized in this chapter.

3.1 Traditional Development

The term traditional development in the gaming industry is used when describing games produced by major companies in the industry. Squaresoft, Ubisoft, Bethesda and Blizzard are currently leading the RPG game industry. However, when inspecting the industry from the gamers' point of view, majority of games produced by the respected companies produce similar and predictable content. Traditional development companies tend to follow the most selling trend in gaming. Therefore, majority of games developed are shooting games. Research conducted by Entertainment Software Association (hereinafter ESA) in 2014, 8 out of 20 most selling games were shooting games (ESA 2015). Of the eight most sold shooter games, four were by one developing company. Treyarch development company has received a high amount of criticism from gamers along the years for producing similar and predictable games. The development problems producing similar products is the concurring problem to multiple development companies. However, the problem is simply the result of industrialized game development field.

Traditional development process compared to indie development is much more restricted from the developer point of view, developers are often just instructed to follow guidelines given by designers. Since the development process is more systematic and planned, products are developed in faster speed and follow strict quality throughout the whole process.

3.2 Indie development

The term indie development is used when describing games developed by small teams or individual developers, significant financial support from publishers or other sources is common within Indie developers. Traditionally, indie development process takes much longer compared to traditional development processes, and the lack of financial aid highly impacts the time that developers can invest in the development. The major difference in Indie development and traditional development from the development point of view comes from the motivation towards the project. The traditional development companies aim strictly to gain most profit from the finished projects, Indie development motivation differs for every developer. The majority of the indie development games however are motivated to publish their projects from the personal passion to the industry. (VanEseltine 2015.)

Majority of the games developed by indie development teams are quite small compared to traditional development games, however indie developers tend to focus on replay value rather than length of the one gameplay of the game. Comparing indie development and traditional development game sales, huge differences between these games is present when looking at the most sold games on the planet. From the 5 most sold games on this planet, only one game is indie development game (Tassi 2016). Game developed by Markus Persson was community driven, therefore the game became just like majority of the gamers desired it to be (Goldberg & Larson 2013).

While Indie development is getting more popular among developers and gamers, traditional development companies do not seem to be interested in Indie development. Unless companies are not interested in buying the project developed by indie developers, traditional companies tend to stay in their own development strategies. Indie developers in general are not interested game companies from the business point of view. However, possible cooperation with game development companies is possible when the game is in the publishing phase. Even though unusual for traditional development companies aiding indie developers in publishing, the possibility is not unheard in publishing. (Pile 2012).

3.3 Advantages and Disadvantages of Indie Development

While Indie development is seen to be the more developer friendly method, advantages following indie development usually cover the cost of disadvantages. Indie development is considered to be more developer friendly method compared to Traditional approach, although more open and accepting towards new ideas and concepts, Indie developers suffer considerable amount when considering funding when developing and publishing games. Lack of funding causes extended development time, lack of motivation, publishing problems and lack of testing for finished products. Since majority of traditional development companies already have their own testing teams and software, Indie development usually lacks these facilities and personnel doing these tests. Indie developers usually must buy testing software and services from other organizations, which can be costly depending on the scale of the game. Although, the majority of Indie games get funding from other sources than their own bank accounts, such as investors and Kickstarter, the cost of developing a complete game is also costly to developers usually requiring personal investment to project also. (Watsham 2013).

Planning projects is considered the hardest part of the Indie development. The most crucial part of the work must be taken seriously or the ending result of project can lead into disaster without a game. When the planning phase of the project is conducted well, vision, technology, design and art style have been chosen, the developers must set to certain timetable for milestones and release. Traditional development companies follow different paths, while designers, investors and directors define time necessary for the project to be concluded. While work load from developer point of view in Traditional development is distinctively smaller when compared to Indie development, developers suffer from the lack of possibility to affect the designed game. Indie development being more open, enables the possibility for developers to affect the designed game without conflict between the designers and developers. (Watsham 2013).

Publishing games as an Indie developer is always a disadvantage compared to traditional development. Even though, possible to publish games as an Indie developer and gain a considerable amount of recognition among gaming community, traditionally large companies can finance large advertisement campaigns on multiple platforms where Indie developers are only able to focus on social media platforms. While traditional development companies spend considerable amounts of money to advertisement campaigns, usually developers themselves have almost no impact towards the campaign. Traditionally developers have no understanding toward advertisement. However, when development team is not closely connected to marketing team major issues can occur when publishing. Major incidents occurring when the development team and marketing team are not communicating is rare, however during the past few years lots of controversies circling around Hello Games publishing of “No Man’s Sky”. Incident concerning “No Man’s Sky” led to an investigation of Hello Games for false advertising, even though in the end allegations were dropped out as it was clear to the gaming community that a connection between developers and the marketing team was missing. (Crecente 2016).

Finally, a major disadvantage Indie development is facing when compared to traditional development is in the hardware department of every developer. Majority of the tools and software used in game development are costly, and understandably not every developer can purchase everything necessary for the development project. Majority of Indie developers therefore tend to favorite open source and free software when developing their projects. Even though, software used by Indie developers are capable in creating same functionalities as paid software, majority of the most popular development tools are costly and better quality compared to open source software. Traditional development companies can invest considerable amounts of money in development tools and hardware during every development process. Therefore, better working environment and hardware are guaranteed in traditional development.

While Indie developers are facing multiple disadvantages when comparing to traditional development companies, majority of Indie developers choose to work without restrictions from outside sources. Majority of Indie developers develop games from their own passions rather than money, which is a major difference when comparing to traditional development companies. Although majority of indie games developed suffer from the lack of popularity, some of these games have become the most profitable games ever.

4 PLANNED FEATURES AND OBJECTS

This chapter discusses and argues for the designed features for this project. Additionally, source information for the design of the features is dealt with in this chapter. Reasoning behind every choice is also discussed in this chapter.

4.1 Character Movement and Actions

Designing character movements and actions depends heavily on the necessary functions required by the designed game. Due the nature of the project, we have chosen to focus on two-dimensional design. When designing and producing these functions, the game type mainly defines the required functions, and with basic “sidescrollers” the amount of required functions can be rounded to minimum. Jumping, shooting and movement controls would be enough. However due the passion to the industry and experience in programming in general, the decision of creating more complex game was decided. The main functions given to “main” character were movement in two horizontal and vertical direction, interaction with objects inside the game environment and attacking non-playable characters (hereinafter NPC).

4.2 Enemy Design and AI

The enemy design is heavily dependent on the player controlled character, enemy design cannot be created much more complicated compared to player controlled character. Enemies have been designed to be simple of nature and easy to edit and add to the game content, this way the amount of development flaws has been minimalized during enemy development phase. The enemies inside the game have been assigned their own “weight”, boundaries and durability, this way player cannot simply walk over these objects without any effect on player controlled character.

Simply assigning enemies to work in same layer as player movable character and immovable object within environment allows them to have same physical laws compared to others within the same layer. Since NPC cannot be controlled by player, AI had to be defined for these objects. Rather than just mindlessly roaming around the game environment, NPC will act only when player controlled character is within viewing distance. NPC will follow tasks assigned to them during the development phase.

4.3 Environment Design

Environment is designed to be as basic as possible with objects restricting player and enemy movement to certain directions. Environment is designed to have details such as shadows effected by light sources, immovable objects and boundaries for player controlled character. Environment design does not have huge impact within the development phase, only objects designed inside the environment have impact on development phase. Environment design is easily changed and aesthetic objects can be added simply even after every other development step has been finished.

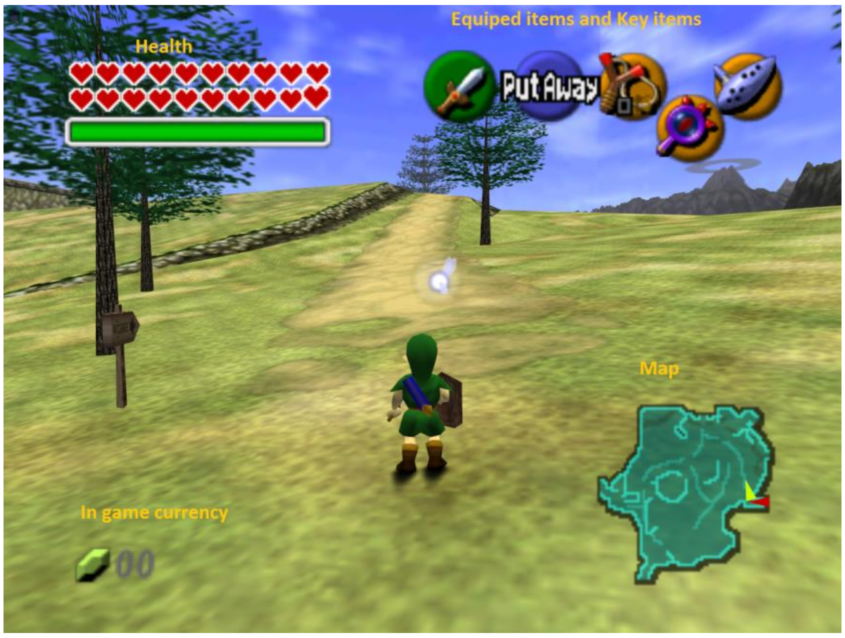
4.4 Camera Control

Camera control for any game is crucial part, without camera controls or specifications it is absolutely impossibility to play the game. The camera behavior depends highly from the design of the game, 3D games often require first person view or third person view. 2D games often follow only third person view which makes designing camera controls for the game much simplistic and easier to manage, due to nature of game designed into this project the camera control follows the main character from third person point of view.

Camera controls are designed to be unable to control specifically by the player, rather move automatically according to player’s moves with controlled character. This function can be assigned directly from unity from camera settings, by connecting camera to follow player movable object camera follows automatically players controls. (Burton 2016)

4.5 Graphical User Interface

Graphical user interface (hereinafter GUI) design and functionality will revolve around simplicity and user friendliness. Titles screen, options and character information during game play are functions designed and necessary for this project. Quoting Albert Einstein “Make it simple, but no simpler”, the key factors of GUI design will follow this strictly. Since many of the games in same genre as designed game, the chosen GUI will resemble closely to these designs. Legend of Zelda: Ocarina of Time GUI resembles a perfect example of interface designed to this project, simplicity and functionality being the main factors of this design



Picture 1. Example of GUI Design Similar to Designed Game Genre

As seen from Picture 1, the design is simplistic yet informative. Following footsteps of Gamesplusjames, guidelines provided by him in his YouTube series will be used as an example and base of the design. Previously mentioned GUI example follows the 8-golden rules proposed by Ben Shneiderman, universal usability, reducing short-term memory load and keeping user in control are the main focus of the design (Shneiderman 2010). Picture 1 pictures a working UI design and is therefore chosen for the development work, known as non-diegetic design. UI is never connected to game world, always seen by the players and informative display eases the connection between the player and the game. (Stonehouse 2014.)

4.6 Sounds

Sound design chosen for this project would follow similar effects within the genre chosen for this game. Due to huge availability of online libraries providing sound effects to be used free in personal projects, choice of recording own sound effects for this project was neglected. Sound effects will be assigned as last part of this project if necessary, since this project is a prototype not every sound effect will be assigned for this project. Music within the game can be assigned directly from Unity interface, therefore scripting to have sounds within the game is not required.

Sound effects can be assigned the same manner as assigning player controllable object actions, simply connecting sound effect to certain action can be used in order the sound effect to work. Sound effects can also be assigned to object immovable by the player, but similar matter these sounds are connected to objects behavior.

5 IMPLEMENTATION

Designed features for this development work are implemented to prototype version in this chapter. Furthermore, software used during the implementation is mentioned in this chapter.

5.1 Methods

In order to create designed functions, guides provided by Unity development team and individual developers were used to construct scripts required to reach the desired goal. Information gathered from various developers' guidelines and support provided by Unity Reddit community were used in case of programming errors and troubleshooting during the development process. Gamesplusjames development diary recorded to YouTube provides plenty of information regarding to designed functions chosen for this project. However, small changed into original script has been made for the script work in desired manner.

5.2 Character Movement and Actions

Generally player controlled character scripting is the easiest phase of development, the hardest part of these functions are controls and animation sequencing. The basic controls of pressing left to go left and so on, the script is simple and easily produced in couple of minutes. Physics of the character were created using Unitys own script called "Rigidbody", the same function will be used later when implementing enemies to the game. Script "PlayerController.cs" is created to have functionalities appropriate to design, script before and after Rigidbody implementation can be seen from table 1 and table 2.

Rigidbody allows to assign certain characteristics to objects inside the game, mass, force and collision are main characteristics required for the developed character to work. As illustrated in table 1, Rigidbody function is assigned to player movable objects by default. Assigning player controlled character mass, allows the character to have momentum in his movement, this gives realistic behavior to objects.

Table 1. PlayerController.cs script, Rigidbody Enabled

```
if (Input.GetAxisRaw ("Horizontal") > 0.5f || Input.GetAxisRaw ("Horizontal") <
-0.5f)
myRigidbody.velocity = new Vector2(Input.GetAxisRaw(“Horizontal”) *
moveSpeed, myRigidbody.velocity.y);
if (Input.GetAxisRaw ("Vertical") > 0.5f || Input.GetAxisRaw ("Vertical") < -0.5f)
myRigidbody.velocity = new Vector2(myRigidbody.velocity.x,
Input.GetAxisRaw(“Vertical”) * moveSpeed);
```

In this form the objects assigned with this script follow basic physics laws, however the script had to be changed to give desired characteristics to objects. Momentum of moved objects had to be changed in a way where when controlling of the object stops, the animation and object stop exactly on the spot. As seen from a table 2, Rigidbody was removed completely from the script to have correct functionality.

Table 2. PlayerController.cs, Rigidbody Removed From the Script

```
when(-0.5f<Input<0.5f)
if(Input.GetAxisRaw(“Horizontal”) < 0.5f && Input.GetAxisRaw(“Horizontal”) >
-0.5f)
{               myRigidbody.velocity = new Vector2 (0f, myRigidbody.velocity.y);
                }
if(Input.GetAxisRaw(“Vertical”) < 0.5f && Input.GetAxisRaw(“Vertical”) > -0.5f)
{               myRigidbody.velocity = new Vector2 (myRigidbody.velocity.x, 0f);
                }
```

Simply assigning these functionalities were enough for player controllable character, further scripting to create more functions for player movable character were unnecessary. Player movable character is not desired to have multiple functions at this point of time, however since further development towards this project is possible, the script has been created to be modified easily. (Gamesplusjames 2015b.)

5.3 Enemies and AI Implementation

On a basis of basic design and functions required from enemies in designed game, the enemies do not differ from player moved character almost at all. The difference comes from making enemies act without players' interaction with them, this is where AI design is crucial. Since every movement enemies have been designed to do and are not effected by players' controls over controllable character, a script was created for enemies to behave certain way. Mainly NPC enemies have been created to guard areas and create difficulty for player when roaming around the environment. The enemies have been designed to hurt player movable character in contact, this is where enemy NPC rigidbody comes in use. When enemy NPC is in contact with player character, they can push them due to higher weight value assigned to them. Although enemy NPC can push player controlled character back, when they are attacked they are pushed back due to the rigidbody function. Assigning these functions to enemy NPC is like scripting player movable character, however since there is no get function inside the script, every function must be scripted individually to have every function desired for enemy NPC. (Unity technologies 2016c).
As seen from Table 3, random variable was created to enemy NPC script to have behavior unexpected by the player. Also, seen from table time of every movement is also assigned inside the script.

Table 3. EnemyController.cs

```
void start()
{
myRIgidbody = GetComponent<Rigidbody2D>();
//timeBetweenMoveCounter = timeBetweenMove;
timeBetweenMoveCounter = Random.Range (timeBetweenMove * 0.75f, timeBetweenMove * 1.25f)
//timeToMoveCounter = timeToMove;
timeToMoveCounter = Random.Range (timeToMove * 0.75f, timeToMove * 1.25f);
}
```

Time between every movement done by enemy NPC is controlled by function “bool moving”, this function is always looking for enemy NPC for movement. When bool moving detects enemy NPC moving it starts the scripts time counter as seen from Table 3, bool moving can be assigned directly from Unity features. (Gamesplusjames 2015c)

5.4 Environment Implementation

Implementation of environment was done mostly with Tiled2Unity software, using free resources provided by opengameart.org. Tiled2Unity was used to create layering to environment, bottom layer is used as a base for movable objects where in direct contact with base layer can be moved. By adding additional layers, objects within the game can be created which can cause collision between moving objects and layers therefore stopping objects. After creating this layering with Tiled2Unity, software can simply create file compatible directly with Unity. When importing environment to Unity, collision laws must be assigned to different layers. When Tiled2Unity is used, automated files are created in the process. Scripts named “ImportTiled2Unity.X” files are all automatically created, however small changes are made into the script changing behavior and class of the script. Multiple layers and objects can be assigned to single file, e.g. all player controlled character objects are connected to “ImportTiled2Unity.cs” script, although when importing single file through Tiled2Unity multiple files are created.

When editing environment data using unity, different layer levels can be assigned to behave certain ways. However, when assigning different layer levels, layer with the highest assigned level will always cover layers assigned under them. Highest layer is used to adding details to environment, this also is used to add collision to highest layer creating objects unmovable and completely unaffected by player controlled character. Environmental implementation requires the least scripting during the whole development process, however layer sorting and creation is time consuming and tiring process. (Henley & Johnson 2014.)

5.5 Camera Control

Unity provides every possible function for camera to behave in desired matter, however it is developers' duty to assign the behavior. Default settings for camera controls would not follow player controlled character, assigning the camera controls to just follow player controlled character can be assigned simply by connecting camera view to player controlled character. Camera follows player controlled characters' movement, this way the camera controls are not directly controlled by the player. (Unity technologies 2016d.)

However, during this project some functionalities of the camera had to be changed for it to behave in desired manner, firstly the camera should not follow the player controlled character at the same speed as the character moves. This is done by adding small delay in the camera controls, also camera movement are limited in a way where camera cannot show objects outside the boundaries of the environment. Simply when player reaches the edge of the environment the camera stops moving in the direction where player is moving the character. Camera also works as trigger for enemies in this development process, when enemy NPC is not in the reach of the camera they remain idle and do not have any functions. Main functionalities of camera will be found from "CameraController.cs", however this file only seeks the controls from Unity platform. Triggering points for camera loading can be found from this file, "Destroy" functionality mentioned in script controls all the objects loaded at the screen simultaneously. This is done to have lighter processing requirement from the game, rather than loading everything at once the game engine is loading only parts of the environment. (Burton 2014.)

Even though all functionalities are provided by Unity, automated script will be defined for the controls. Illustrated in Table 4, camera controls have been assigned to follow game object.

Table 4. CameraController.cs, Automatically Created Camera Controller Script

```
using System.Collections;
using UnityEngine;
public class CameraController : MonoBehaviour{
public GameObject followTarget;
private Vector3 targetPos;
public float moveSpeed;
private static bool cameraExists;
void Start(){
if (!cameraExists){
cameraExists = true;
DontDestroyOnLoad(transform.gameObject);}
else{
Destroy(gameObject);}
}void Update(){
targetPos = new Vector3(followTarget.transform.position.x,
followTarget.transform.position.y, transform.position.z);
transform.position = Vector3.Lerp(transform.position, targetPos, moveSpeed
* Time.deltaTime);}}
```

As seen from Table 4, the script also defines camera speed, positioning and loading trigger. Even though the script is short and simplistic, it has all defining factors for it to work in desired manner. Functionalities of camera should not be developed in directly by adding functionalities to the script, since Unity does not update the camera functionalities automatically if not directly done so in Unity. Functionalities to camera behavior can be added from camera settings without having major effect to other parts of the game.

5.6 Animation

During the development of this thesis work the animation and graphical aspect of the game did not play a huge part. During the project, already existing online libraries providing objects and graphical designs were used to test everything to work in order. Some cases in this work some objects could not be found from free online sources, these animation objects were created in Photoshop and afterwards imported using Tiled2Unity. Animation for objects is simply done using sequencing of different states of same graphical object. As seen from

Figure 5, one object has multiple different states and they are used every time objects behave certain way.

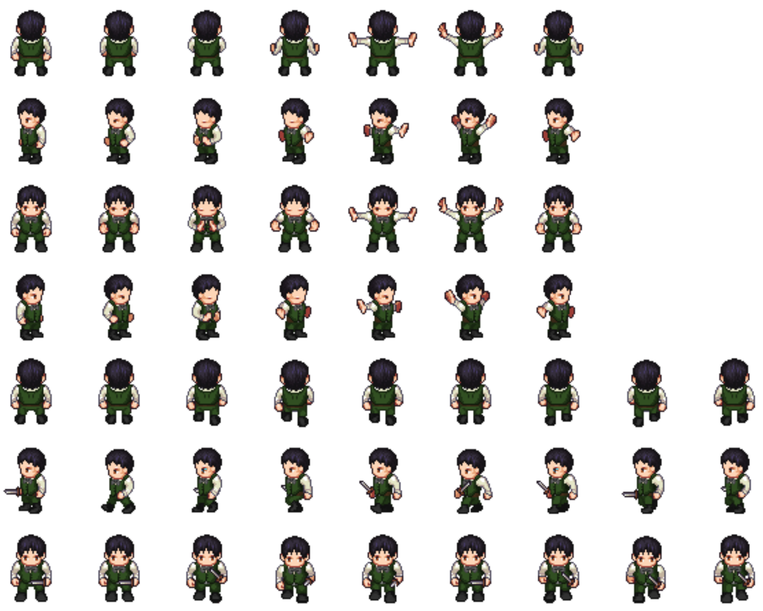
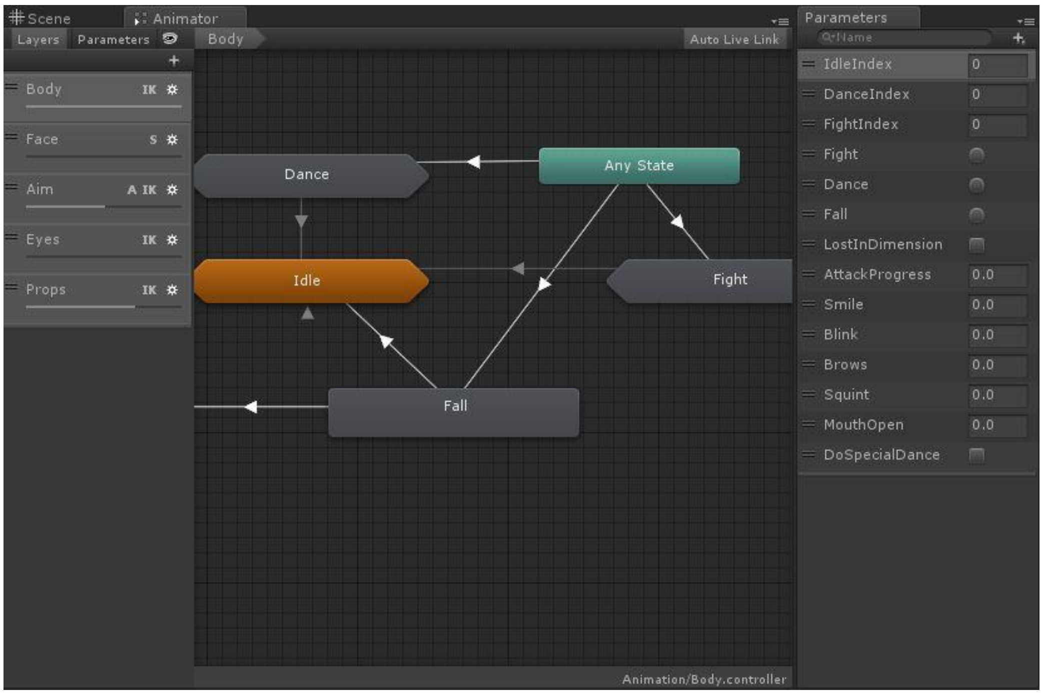


Figure 4. Player Movable Character Animations.

Sequencing of animation does not require any scripting, however lots of work must be done to have working animation. During the development of this thesis work, lots of difficulties with animation sequencing were encountered. Troubleshooting for animation sequencing had to be done all by hand, Unity does not provide any features to help finding flaws in the development work of animation. Although lots of difficulties were encountered, all flaws were corrected in the end by starting the animation work from scratch after the first few failures.

As seen from the Figure 6, the animation matrix for player controlled character does not look too complicated.



IdleIndex	0
DanceIndex	0
FightIndex	0
Fight	●
Dance	●
Fall	●
LostInDimension	☐
AttackProgress	0.0
Smile	0.0
Blink	0.0
Brows	0.0
Squint	0.0
MouthOpen	0.0
DoSpecialDance	☐

Figure 5. Animation Sequences in Player Controlled Character

Although this matrix seems simple and easy to understand, lots of research had to be done to have correctly behaving animation. As seen from the figure above idle state is the starting and ending point of every animation, this means that after object is no longer controlled nor effected by anything within the game environment this should be the default animation state during that time. Main problem concerning the developers design of the animation sequencing came in the form of interaction between the player controlled character and enemy NPC, due the lack of connection between player controlled character and NPC animation they did not have any reaction towards each other from the animation point of view. Every animation in this project react with script “TileAnimator.cs”, this automatically created script lists automated animations of all objects within the game. “StartTime” and “Duration” controls the length of animation, ending point is not defined in this script to have desired behavior for enemy NPC since player controlled character does not have “StartTime” or “Duration” function defined in the script.

Even though the console of Unity gave information of interaction between player controlled character and NPC characters, animation was unaffected by this. Due the problems with the animator in the beginning of this part of the development work, this graphical aspect was hard to be fixed without any outside sources. Luckily to our chosen guide for this development work, Gamesplusjames has excellent guide how to work with animator of Unity. The problem within our development work was corrected by adding trigger points for certain animations when any of the deciding factors would be true. When player controlled character has interaction between enemy NPC, the enemy NPC causes damage to player character, this is used as one of the triggering points for animation. When player controlled character is touched by the enemy NPC the animation of the player controlled character flashes, for this part new animations were not needed to be created and this function was found from Unity animator. (Strout 2015.)
Although animation development part of this thesis work took so much time, the developers are not satisfied to the result. Every part of the animation works as wanted, but room for improvement for the sequencing and timing is required in order to finish the end result of the game.

5.7 Graphical User Interface

GUI developed for this project is highly similar to the interface illustrated on the Figure 2. The GUI build for this project focuses more on functionality than any other aspects, the GUI behaves as a layer and object inside the game all at the same time. This is one reason why GUI development part of this work did not have any new obstacles concerning other parts of the work. GUI behaves similar to other game objects within the game, change happens when certain trigger actions happen. Enemy NPC contact, picking up items, change in the map and quest updates, all these parts have effect on GUI. Design part of the GUI follows strictly the design found in similar games in this category, during the development phase the aim was to create as simple as possible design. As seen from Figure 2, the design can be simple yet include lots of knowledge concerning the game. Because the GUI plays a small part in this project, the further development is not necessary for this project. (Alismuffin 2011.)

GUI also includes functional part to it, player is able to control the chosen weapon and item that they have collected. Some functions developed during the scripting phase have been connected to the GUI, however the changes happening to the GUI does not have any effect to the game, only to the graphical changes are happening inside the GUI.

5.8 Sounds

During the animation development phase, the developers focus did not expand to the sound effects and music. The sound effects can be attached to animations directly, or they can be set up to be played when certain triggering effects are happening in the game. Online libraries of free sound effects were used in this development part of the thesis work, therefore recording own sound effects was not required for this project. During the sound implementation, the developers ran into problem with the difference in time between the animation and sound. Because the sounds and animation length time wise was different, already existing sounds did not match the sound effects available online. When the sound development process started, everything began from testing how to attach certain sounds to certain animations. The main problem was that sound effect would start immediately when moving, rather than making step sounds when feet hit the ground, the stepping sound started when the feet got off the ground. (Uccello 2016).

The problem with sounds starting from wrong points was corrected using Audacity software, small tweaking to the sound effects was done in order to have their starting points to have same as the animation. Ending of the sound effect can be defined to end exactly when the animation ends, or to have them played completely after the ending of the animation. Sound implementation for this thesis work was considered even from the beginning as the least important feature. Most important sound effects were assigned to the development work, majority of the game sound effects were left out from this development work.

6 TESTING

The testing phase is crucial to every software development process. In this chapter, the functions and scripts are tested for bugs and flaws implemented during the development phase. Improvement of existing features were implemented during the testing phase. Additionally, research for future development of this thesis work will be defined in this chapter.

6.1 Troubleshooting and Testing

Since the best way of finding flaws and bugs inside game is by playing it, the developers assigned friends to test the end result and report possible flaws inside the game. People chosen for this process were chosen from close by friends of the developers who have history in gaming and in game development, although these people have experience in game development they did not have experience in game testing. During the testing phase, test subjects were given instructions to play the game normally at first and following small guidelines provided by the developers. The testing phase also included tests done by the developers, this way the game had tests from black box and white box testing. (Software Testing Fundamentals 2016a).

6.2 Difference Between Black and White Box Testing

The testing phase of this project was divided into two main phases, people who are not familiar with the design, structure and implementation of the designed software will be taking part in black box testing. People who are familiar with everything done for the software will be taking white box test, usually people who conduct the white box testing are the developers of the software, like in this thesis work also. (Software Testing Fundamentals 2016b.)
Black box testing is used to get more in depth view towards the system and its design, this is a great way of finding suitable solutions for existing flaws within the design. Test subjects do not require any programming knowledge to evaluate the product, this way the focus is more on the output of the program

rather than the internal mechanism of the software developed. During black box testing it is crucial for the testers to have no access to source code, the only concept defined for the testers should be the expected outcome. The test should return results from difference between expected outcome and actual outcome, however black box testing can be considered redundant if the developers have run the testing before the test subjects. (Williams 2006a).

White box testing is considered more in depth testing of the software, since the software design, implementation and structure are already familiar to the test subjects' source code will be provided during this phase. During white box testing the subjects were testing individual cases of the designed software, rather than focusing on the large view of the solution. End result of white box testing should find internal flaws from programming flaws to design flaws, since the test subjects are required to have previous knowledge towards programming and design of the software test subjects can be divided to focus on different parts of the software. (Williams 2006b).

6.3 Black Box Testing in Case Development Work

The black box testing begins from designing suitable plan for the test subject, since this test does not require any programming knowledge from the test subjects close by friends without programming experience were chosen for this part. The test was designed for subjects to gain view from mainly of the game design and functions, this was done by sending asking the test subjects to walk around the environment of the game and encounter the first enemy NPC. The test was focusing on the players point of view, this way developers would get more in depth view of the design flaws and desired functions. (Software testing fundamentals 2016c.)

The test was concluded in one session with the test subjects, as seen from the following tables, the black box test was divided in to three major categories. During the black box testing software testing fundamentals were used to create the tables, this procedure is typical for any software testing procedure. (Software testing fundamentals 2016d.)

As seen from table 5, GUI was the first aspect of development tested in this phase. Even though, the GUI development was successful from test subject point of view, further development is necessary when game is developed further.

Table 5. Black Box Testing Procedures for GUI Testing

Features	GUI
Requirements	Game demo and platform
Test procedure	1. GUI assessment and testing when moving around the game environment
Expected results	GUI reacts to test subjects actions
Results	GUI reacted according to the programming
Test status	Success
Notes	GUI reacted to player controlled character movement and actions
Test date	December 20, 2016

During the testing phase of GUI, test subject pointed out the minimalistic design and functionalities. Even though, working GUI is success for this project, lacking functionalities were mentioned by the test subjects.

Following table consists test results from player controlled character controls and animation, simplistic design of the planned features was expected to provide excellent results of this test.

Table 6. Black Box Testing Procedure for Player Controlled Character Controls and Animation

Features	Player controlled character controls and animation
Requirements	Test subject controls the player controlled character
Test procedures	1. Test subjects control the character around the game environment 2. Explore the controls and access the animations
Expected results	Character animation and movement is according to developed features
Results	Character animation was flawed and movement was according to developed controls
Test status	Partial success
Notes	Character moved perfectly according to designed controls, while moving left the character had animation while moving right.
Test date	December 20, 2016

As seen from table 6, expected results were not met during this phase. Even though, the flaw occurring during the testing phase was minor failure. However, correction towards the animation flaw was corrected easily and animation was considered success after the correction.

Enemy NPC testing for this development work was done by using only one enemy character. Even though the test was small in order to test enemy NPC, all necessary information was reached during the tests.

Table 7. Black Box Testing Procedure for Enemy NPC Encounters

Feature	Encountering enemy NPC
Requirements	Player encountered Character has contact with enemy NPC
Test procedures	1. Test subject moves player controlled character to enemy NPC
Expected results	Player controlled character reacts to enemy NPC and initiates correct animation
Results	Player controlled character reacts to enemy NPC and correct animation is played
Status	Success
Notes	When player controlled character meets enemy NPC, player controlled is harmed and correct animation is played.
Test Date	December 20, 2016.

As seen from table 7, the scripted functions for enemy AI were success. While features tested in this phase were working and according to design, further development towards enemy AI is desired in order to create complete game.

During the black box testing procedures most of the features were found to be successfully developed, working according to design and having correct reactions. However, during the testing phase animations proved to have been including flaws and had to be corrected after the black box testing. Majority of the tests done in white box testing cover up the test procedures left out from black box testing, camera controls, sound implementation and environment testing is taken into more in depth view during the white box testing.

Although Majority of the features tested during the black box test were successful and liked by the test subjects, the feedback given by the test subjects left space for improvement. Animation of the characters seemed bit stiff and incorrectly timed, the GUI had information connected to it that did not have any use in the test version of the game and enemy NPC reaction range was proven to be too big compared to the size of the object in game.

Black box testing was executed by using two test subjects, JESSH91 and Mäksä asked to remain under their online alias during the test. The test subjects have extensive experience in gaming and large amount of knowledge towards the development work game genre.

6.4 White Box Testing

White box testing procedures are highly more sophisticated and accurate compared to black box testing, although the black box testing reveals flaws within the game, it cannot explain the reason behind them. The reason why white box testing was also used as a part of this project, is to find flaws from the programming point of view and resolve these problems during the testing. While white box testing procedures were divided into different parts similar to the development process, flaws encountered during the development phase could have been listed to white box testing procedures. However, the end result of this development work would have extended far too much and therefore was left out.

As seen from the following tables, the white box testing procedure has more in depth view of different aspects of the game. Animations, enemy AI, camera controls, sounds, environment and player controlled character controls were taken into the test and occurring flaws were corrected after the test. Suggested features were not implemented during this phase. The following tables are developed from Software testing fundamentals guide. (Software Testing Fundamentals 2016e).

During white box testing, same functionalities tested during the black box testing were conducted using new test subject. As seen from table 8, GUI was also taken as a first testing subject. However, during the GUI testing, emphasis was on functionality rather than design.

Table 8. White Box Testing to GUI

Feature	GUI design and features
Requirements	Unity platform and the development case work
Test procedure	1. GUI design and features are tested for connectivity and reaction to game environment 2. Scripts connecting player controlled character and GUI are reviewed and tested 3. Connection between GUI and other layers is tested
Expected results	1. GUI features are working according to the design 2. Player controlled character actions have effect to GUI 3. GUI does not have any physical connection to other layers
Results	1. GUI has implemented features that are not connected to anything 2. Player controlled characters actions, i.e contact with enemy NPC has effect to GUI 3. GUI is not connected to other layers
Status	Partial success
Notes	Game currency feature in GUI is not connected to any feature within the game. Other aspects worked according to the design, room for improvement detected.
Test Date	December 21, 2016

As seen from the table 8, the white box testing is more sophisticated and follows more developers point of view rather than players point of view. White box testing is more time consuming and can reveal the same results as black box testing, but when using both tests equally can reveal shortcomings from players and developers point of view.

Animation testing during white box testing was seen as a necessity for this project, considerable amount of time was invested in this development the results were expected to be successful. As shown in table 9, during the test character controls are not taken into account during animation testing like in the black box testing.

Table 9. White Box Testing for Animation

Feature	Animation
Requirements	Unity platform, development case work and test subjects control the player controlled character and inspect the animations
Test procedure	1. Test subjects controls the player movable character and inspects the animations 2. Animations other than player controlled character are tested 3. Reaction between different animations is tested
Expected results	1. Player controlled reacts to controls are initiates the correct animation 2. Enemy NPC animation works similar to the player controlled character 3. Animations have correct reactions to other animations
Results	1. Player controlled character animation while moving to right has the animation while moving left 2. Enemy NPC has correct animation 3. Some animations lack the reaction to other animations
Status	Partial success
Notes	The player controlled character animation while moving right was never connected to the function of moving right. Some implemented animations lack the reaction to other animations.
Test date	December 22, 2016

As seen from table 9, animations occurred to be returning problem for this case development work. Flaws occurred during the animation test were dealt with when they occurred during the testing, this extended the time required for every test. Correcting flaws from animations could take a tremendous amount of time, from ten minutes to one day, however the ends result is worth the time invested in this process.

The following table is describing the sound implementation white box testing, because this process could have been done by black box test subjects rather than the white box test subjects, the end result is not satisfactory for the developers point of view.

Table 10. White Box Testing of Sound Implementation

Feature	Sound implementation
Requirements	Unity platform, access to development work and source code
Test procedure	1. Player controlled character is controlled and tested for correct sound effects 2. Enemy NPC is tested for sound effects 3. Background music implementation is reviewed
Expected results	1. Player controlled character reacts to all functions and correct sound effect is played 2. Enemy NPC has similar reaction as player controlled character 3. Background music plays and loops after ending
Results	1. Player controlled character did not have sound effects connected to every function 2. Enemy NPC is lacking all sound effects 3. Background music does not loop
Status	Failure
Notes	Player controlled character did not have sound effects when walking right, interacting with enemy NPC or when the player dies.
Test date	December 23, 2016

As seen from table 10, major issues from sound implementation was encountered during the white box testing phase. Problem did not occur during the black box testing, sounds for the game were disabled during the black box testing. Major improvements were discovered from the sound implementation phase, due the lack of experience with Unity platform in general, the development process after white box testing took longer than expected. The process of further sound development was neglected at this time and will be dealt with in the future.

The following table is focusing on the camera controls and implementation of the case development work, this test was left out from black box testing phase because camera behavior seemed correct at that time.

Table 11. White Box Testing of the Camera Controls

Feature	Camera controls
Requirements	Unity platform, access to development case work and source code concerning camera controls
Test procedure	1. Move player controlled character to every corner of the game environment 2. Test reaction of sudden changes of character movement
Expected results	1. Camera has small delay when following player controlled character 2. Camera stops moving when meeting the end of the game environment 3. Camera follows player controlled character when changes in direction occur
Results	1. Camera has small delay and follows player controlled character where every it is controlled to 2. Camera continues moving even when player controlled character find obstacle stopping the movement 3. Camera has problems following the player controlled character when sudden change in direction occurs
Status	Partial success
Notes	Camera scripting requires more development and boundaries for camera movement has to be further tested.
Test Date	December 24, 2016

Tests done to camera controls found multiple problems that did not occur during the black box testing, possible reason for this result might be due the instructions given to the test subjects. Although camera controls did not meet the requirements set up to it during the development phase, small adjustments after the testing phase corrected most of the flaws. Boundaries set up to the camera still has flaws in it and require extensive development work in the future.

Enemy AI was developed for this project to give hostile NPC to behave in desired matter in this game. The enemy AI cannot be seen during the game play, therefore testing the AI from the backend during the testing phase was necessary. The table 12 shows the testing procedure for enemy AI.

Table 12. White Box Testing for Enemy AI

Feature	Enemy AI
Requirements	Unity platform, access to source code and controlling player controlled character with enemy NPC interaction
Test procedure	1. Player controlled character is moved to viewing distance from the enemy NPC 2. Enemy reaction to player is tested 3. Reaction to other objects in game is tested
Expected results	1. Enemy NPC starts moving only when the player controlled character is in viewing distance from the enemy NPC 2. Enemy NPC reacts to player according to the design 3. Enemy NPC changes direction when meeting immovable obstacles or game environment edge
Results	1. Enemy NPC starts moving when in viewing distance from player controlled character 2. Enemy NPC has contact with player and deals damage 3. Enemies do not change direction when meeting obstacles or edge of the environment
Status	Partial success
Notes	Enemy NPC has correct AI functions, reaction to game environment was not enabled during the testing phase.
Test date	December 27, 2016

As seen from table 12, any major flaws from enemy AI was not found. Main flaw found from enemy AI was reaction with other game objects, fortunately this was not found to be too difficult flaw to be bypassed with small tweaking with the enemy script.

The following table is from the last white box testing done during this development work, environment testing consist mostly inspecting different layers used in the game. During the white box testing every flaw found was corrected, also new improvements were implemented during this phase. The table 13 shows the procedure conducted to the game environment.

Table 13. White Box Testing for Game Environment

Feature	Game environment
Requirements	Access to Unity platform and source code
Test procedure	1. Visual inspection of different layers of the game environment, excluding the GUI layer
Expected results	1. Layers do not run into a conflict between different layers 2. Every object is assigned to correct layer 3. Layers have been defined correctly and boundaries have been added to objects
Results	1. Layers have been assigned correctly in desired level 2. Objects were assigned to correct layer 3. Layers are defined correctly, some boundaries between objects was encountered.
Status	Success
Notes	Correcting the boundaries of objects was done immediately when encountered, this required little no time therefore it is not considered as a failure.
Date	December 30, 2016

As seen from the table 13, the environment design was the most successful developed feature of this development work. Flaws occurring during the white box testing of the game environment were easy to be corrected and improved, further development towards game environment is not required at this point. Expanding the game environment is possible in the future development of this development work

White box testing is time consuming phase were many programming and scripting errors can be discovered easily, during this thesis development work the main test subjects were the developers themselves. Both developers conducted their own tests for every category of the white box testing, in the end information was gathered and combined to create these tables.

6.5 Grey Box Testing

The last phase of testing for the developed case work is combination of black box testing and white box testing, this means that the developed work will be presented to a person who has no previous experience in the developed work but has knowledge of the programming and source code will be supplied to the subject. This part of the testing was concluded by a close friend of the developers, however from the test subjects request she wanted to remain unnamed. This part of the testing phase does not include any tables, due to large similarity between the white box testing results and grey box test results it was seen as unnecessary portion for this thesis development work.

Main difference between white box testing subjects and grey box testing subject was the experience using the programming software and sound editing software. (Software testing fundamentals 2016f.)

During the testing phase majority of the flaws occurred during white box and black box testing were discovered by the test subject, further more improvement towards sound implementation was suggested by the test subject. New major flaws were not encountered in this phase of the testing, however small minor flaws in unknown in advance was found from animation. These flaws will be developed further after this development work.

The grey box testing was not done extensively like white box testing, every category was reviewed and tested in five hour period in January 2, 2017. During the testing phase, only one of the developers was present when grey box testing was conducted, this does not affect the end result of the test in anyway.

Main issues occurring during the grey box testing phase were graphical, from developers point of view the scripts are correct and working. However, from graphical point of view there is plenty of room for improvement, since online sources were used for this thesis work project this was expected. When the tests were concluded by the test subject, her input to sound implementation was used in order to create more functioning sound effects. Further cooperation with the test subject is expected from the future.

7 CONCLUSIONS

The objectives of this thesis was to expand the knowledge of the developers towards the game development and indie games. Using other developers' development diaries and blogs to create the development case work played a huge part in this work. At the beginning of this development work, other game development platforms were considered. On the basis of this consideration, Unity was found-to be the best for beginners and experienced programmers. Due to a huge amount of online resources and guidelines provided by Unity company, it would have been an unnecessary step to try to find similar support for other development platforms. Completing a full game from the chosen category was perceived as a too large an objective for this process and, therefore, only a demo version of the game was developed.

Carrying out different stages of this project proved to be totally different from what was expected. A lot less of scripting was required from the developers due to the functions provided by the Unity platform. However, in order to create desired functions and actions for the designed case work, most of the scripts automatically created by Unity were changed to have desired functionalities. The choice of using #C as the scripting language proved to be the correct choice. Even though JS was already a familiar language to the developers, the majority of the online communities is using #C to complete their projects.

This thesis work proved to be much harder than originally expected. The largest obstacle during this project proved to be the Unity platform and computing power of the development tools used in this process. During this project, one of the main development computers used had major complications and the majority of the information stored in it was lost. Additionally, the majority of the developed game was lost at that time. Overcoming these obstacles taught valuable lessons to both programmers, in that one must never underestimate the value of backup save files.

BIBLIOGRAPHY

Alismuffin 2011. Creating custom GUI Skins Part 1. Accessed 10 December 2016
https://forum.unity3d.com/threads/creating-custom-gui-skins-part-one.113055/.

The Audacity team 2016. Open source, cross-platform audio software for multi-track recording and editing. Accessed December 5, 2016
http://www.audacityteam.org/.

Barton, S. 2016. Tiled2Unity: Tiled Support for Unity. Accessed December 2, 2016
http://www.seanba.com/tiled2unity.

Boehm, B., Lane, J. A., Koolmanojwong, S. & Turner, R. 2014. The incremental commitment Spiral Model. Accessed 27 February 2017
http://ptgmedia.pearsoncmg.com/images/9780321808226/samplepages/0321808223.pdf.

Burton, C. 2016. Working with 2D cameras (Unity 2D) Accessed December 6, 2016
http://adventurecreator.org/tutorials/working-2d-cameras-unity-2d.

Crecente, B. No Man’s Sky creator cleared of false advertising allegations. Accessed February 20, 2017.
https://www.polygon.com/2016/11/30/13791782/no-mans-sky-false-advertising-results.

Entertainment software association 2015. Essential facts about the computer and video game industry. Accessed February 15, 2017
http://www.theesa.com/wp-content/uploads/2015/04/ESA-Essential-Facts-2015.pdf.

Gamesplusjames 2015a. Unity RPG tutorial – Learn to make RPG game and learn C#!. Accessed December 10, 2016.
https://www.youtube.com/watch?v=Pk3GCgaNVTY&list=PLiyfvmtjWC_X6e0EYLPczO9tNCkm2dzkm.

Gamesplusjames 2015b. Unity RPG tutorial #11 – Making enemies Accessed December 9, 2016
https://www.youtube.com/watch?v=d3lhb1y_89U&list=PLiyfvmtjWC_X6e0EYLPczO9tNCkm2dzkm&index=11.

Gamesplusjames 2015c. Unity RPG tutorial #2 – Player movement. Accessed December 8, 2016.
https://www.youtube.com/watch?v=Tm2L-_0eIeY&list=PLiyfvmtjWC_X6e0EYLPczO9tNCkm2dzkm&index=2.

Goldberg, D. & Larsson, L. 2013. The Amazingly unlikely story of how Minecraft was born. Accessed February 20, 2017.
https://www.wired.com/2013/11/minecraft-book/.

Henley, J. A. & Johnson, M. 2014. Learning 2D game development with Unity. Accessed December 10, 2016 http://ptgmedia.pearsoncmg.com/images/9780321957726/samplepages/9780321957726.pdf.

Hunt, L. 2007. C# Coding standards for .NET. Accessed February 25, 2017 https://aspblogs.blob.core.windows.net/media/lhunt/Publications/CSharp Coding Standards.pdf.

Lehtinen, L., Junnonen, J. A., Kärmä, S. & Pekuri, L. Accessed March 20, 2017 http://www.gpmfirst.com/books/designs-methods-and-practices-research-project-management/constructive-research-approach.

Notepad++ 2016. About. Accessed December 1, 2016 https://notepad-plus-plus.org/.

Pile, J 2012. The difference Between Indie and Non-Indie Game Developers. Accessed February 12, 2017 http://prof.johnpile.com/2012/07/08/the-difference-between-indie-and-non-indie-game-developers/.

Shneiderman, B 2010. The Eight Golden Rules of Interface Design. Accessed December 7, 2016 https://www.cs.umd.edu/users/ben/goldenrules.html.

Software Testing Fundamentals 2016a. Difference between Black Box Testing and White Box Testing. Accessed January 5, 2016 http://softwaretestingfundamentals.com/differences-between-black-box-testing-and-white-box-testing/.

Software Testing Fundamentals 2016b. Difference between Black Box Testing and White Box Testing. Accessed January 5, 2016 http://softwaretestingfundamentals.com/differences-between-black-box-testing-and-white-box-testing/.

Software testing fundamentals 2016c. Black box testing. Accessed January 11, 2016 http://softwaretestingfundamentals.com/black-box-testing/.

Software testing fundamentals 2016d. White box testing. Accessed January 14, 2016 http://softwaretestingfundamentals.com/white-box-testing/.

Software testing fundamentals 2016e. Test case. Accessed January 20, 2016. http://softwaretestingfundamentals.com/test-case.

Software testing fundamentals 2016f. Grey box testing. Accessed January 20, 2016. http://softwaretestingfundamentals.com/gray-box-testing/.

Stonehouse, A 2014. User interface design in video games. Accessed February 25, 2016
http://www.gamasutra.com/blogs/AnthonyStonehouse/20140227/211823/User_interface_design_in_video_games.php.

Strout, J 2015. 2D animation methods in Unity Accessed December 5, 2016
http://www.gamasutra.com/blogs/JoeStrout/20150807/250646/2D_Animation_Methods_in_Unity.php.
Tassi, P 2016. Here are The Five Best-Selling Video Games Of All Time. Accessed May 15, 2017.
https://www.forbes.com/sites/insertcoin/2016/07/08/here-are-the-five-best-selling-video-games-of-all-time/#330ca2055926.

Uccello, A 2016. Introduction to Unity sound. Accessed January 7, 2016.
https://www.raywenderlich.com/132145/introduction-unity-sound.

Unity Technologies 2016a. 2D Game Development Walkthrough. Accessed December 2, 2016.
https://unity3d.com/learn/tutorials/topics/2d-game-creation/2d-game-development-walkthrough.

Unity Technologies 2017b. Version control integration. Accessed February 27, 2017
https://docs.unity3d.com/Manual/Versioncontrolintegration.html.

VanEseltine, C 2015. Motivation for the Solo Indie Game Dev. Accessed February 15, 2017
http://www.gamasutra.com/blogs/CarolynVanEseltine/20150609/245543/motivation_for_the_Solo_Indie_Game_Dev_with_commentary_by_yayfrens.php.

Watsham, J 2013. Self-publishing vs. having a (traditional) publisher, as told by Renegade Kid. Accessed February 20, 2017
http://indiegames.com/2013/10/self-publishing_vs_working_wit.html.

Williams, L 2006a. Testing overview and Black box testing. Accessed January 8, 2016
http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf.

Williams, L 2006b. White Box Testing. Accessed January 9, 2016.
http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf.

APPENDICES

Appendix 1. PlayerController.cs

```
using System.Collections;
using UnityEngine;
public class CameraController : MonoBehaviour
{
    public GameObject followTarget;
    private Vector3 targetPos;
    public float moveSpeed;
    private static bool cameraExists;
    // Use this for initialization
    void Start()
    {
        if (!cameraExists)
        {
            cameraExists = true;
            DontDestroyOnLoad(transform.gameObject);
        }
        else
        {
            Destroy(gameObject);
        }
    }
    // Update is called once per frame
    void Update()
    {
        targetPos = new Vector3(followTarget.transform.position.x,
followTarget.transform.position.y, transform.position.z);
        transform.position = Vector3.Lerp(transform.position, targetPos,
moveSpeed * Time.deltaTime);

    }
}
```

Appendix 2. EnemyControl.cs

```
public float moveSpeed;
private Rigidbody2D myRigidbody;
private bool moving;
public float timeBetweenMove;
public float timeBetweenMoveCounter;
public float timeToMove;
private Vector3 moveDirection;
void start()
{
myRIgidbody = GetComponent<Rigidbody2D>();
timeBetweenMoveCounter = timeBetweenMove;
timeBetweenMoveCounter = Random.Range (timeBetweenMove * 0.75f,
timeBetweenMove * 1.25f)
timeToMoveCounter = timeToMove;
timeToMoveCounter = Random.Range (timeToMove * 0.75f, timeToMove *
1.25f);
}
Void Update()
{
     If (moving)
     {    timeToMoveCounter -= timeToMove.deltaTime
     myRigidbody.velocity = moveDirection;
        if (timeToMoveCounter < 0f){
        moving = false;
        timeBetweenMoveCounter = timeBetweenMove;
       timeBetweenMoveCounter = Random.Range (timeBetweenMove * 0.75f,
timeBetweenMove * 1.25f)
          }
     } else
          {
          timeBetweenMoveCounter -=Time.deltaTime;
          myRigidbody.velocity = Vector2.zero;
          if(timeBetweenMoveCounter < 0f){
          moving = true;
          timeToMoveCounter = timeToMove;
          timeToMoveCounter = Random.Range (timeToMove * 0.75f,
     timeToMove * 1.25f);
          moveDirection = new Vector3(Random.Range(-1f, 1f), Random.Range(-
     1f, 1f), 0f);
          }
     }
}
```

Appendix 3. ImportTiled2Unity.cs 1(3)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Text;
using System.Xml;
using UnityEditor;
using UnityEngine;

namespace Tiled2Unity
{
   partial class ImportTiled2Unity : IDisposable
   {
      private string fullPathToFile = "";
      private string pathToTiled2UnityRoot = "";
      private string assetPathToTiled2UnityRoot = "";

      public ImportTiled2Unity(string file)
      {
         this.fullPathToFile = Path.GetFullPath(file);

         // Discover the root of the Tiled2Unity scripts and assets
         this.pathToTiled2UnityRoot =
Path.GetDirectoryName(this.fullPathToFile);
         int index = this.pathToTiled2UnityRoot.LastIndexOf("Tiled2Unity",
StringComparison.InvariantCultureIgnoreCase);
         if (index == -1)
         {
            Debug.LogError(String.Format("There is an error with your
Tiled2Unity install. Could not find Tiled2Unity folder in {0}", file));
         }
         else
         {
            this.pathToTiled2UnityRoot =
this.pathToTiled2UnityRoot.Remove(index + "Tiled2Unity".Length);
         }

         this.fullPathToFile =
this.fullPathToFile.Replace(Path.DirectorySeparatorChar, '/');
         this.pathToTiled2UnityRoot =
this.pathToTiled2UnityRoot.Replace(Path.DirectorySeparatorChar, '/');

         // Figure out the path from "Assets" to "Tiled2Unity" root folder
         this.assetPathToTiled2UnityRoot =
this.pathToTiled2UnityRoot.Remove(0, Application.dataPath.Count());
         this.assetPathToTiled2UnityRoot = "Assets" +
this.assetPathToTiled2UnityRoot;
```

```
        }
        public bool IsTiled2UnityFile()
        {
           return this.fullPathToFile.EndsWith(".tiled2unity.xml");
        }
        public bool IsTiled2UnityTexture()
        {
           bool startsWith = this.fullPathToFile.Contains("/Tiled2Unity/Textures/");
           bool endsWithTxt = this.fullPathToFile.EndsWith(".txt");
           return startsWith && !endsWithTxt;
        }

        public bool IsTiled2UnityWavefrontObj()
        {
           bool contains = this.fullPathToFile.Contains("/Tiled2Unity/Meshes/");
           bool endsWith = this.fullPathToFile.EndsWith(".obj");
           return contains && endsWith;
        }

        public bool IsTiled2UnityPrefab()
        {
           bool startsWith = this.fullPathToFile.Contains("/Tiled2Unity/Prefabs/");
           bool endsWith = this.fullPathToFile.EndsWith(".prefab");
           return startsWith && endsWith;
        }

        public string GetMeshAssetPath(string file)
        {
           string name = Path.GetFileNameWithoutExtension(file);
           string meshAsset = String.Format("{0}/Meshes/{1}.obj",
this.assetPathToTiled2UnityRoot, name);
           return meshAsset;
        }

        public string GetMaterialAssetPath(string file)
        {
           string name = Path.GetFileNameWithoutExtension(file);
           string materialAsset = String.Format("{0}/Materials/{1}.mat",
this.assetPathToTiled2UnityRoot, name);
           return materialAsset;
        }

        public string GetTextureAssetPath(string filename)
        {
           // Keep the extention given (png, tga, etc.)
           filename = Path.GetFileName(filename);
           string textureAsset = String.Format("{0}/Textures/{1}",
this.assetPathToTiled2UnityRoot, filename);
           return textureAsset;
        }
        public string GetXmlImportAssetPath(string name)
        {
#if !UNITY_WEBPLAYER
            name =
Tiled2Unity.ImportBehaviour.GetFilenameWithoutTiled2UnityExtension(name);
#endif
            string xmlAsset = String.Format("{0}/Imported/{1}.tiled2unity.xml",
this.assetPathToTiled2UnityRoot, name);
            return xmlAsset;
        }
        public string GetPrefabAssetPath(string name, bool isResource, string
extraPath)
        {
            string prefabAsset = "";
            if (isResource)
            {
                if (String.IsNullOrEmpty(extraPath))
                {
                    // Put the prefab into a "Resources" folder so it can be instantiated
through script
                    prefabAsset = String.Format("{0}/Prefabs/Resources/{1}.prefab",
this.assetPathToTiled2UnityRoot, name);
                }
                else
                {
                    // Put the prefab into a "Resources/extraPath" folder so it can be
instantiated through script
                    prefabAsset =
String.Format("{0}/Prefabs/Resources/{1}/{2}.prefab",
this.assetPathToTiled2UnityRoot, extraPath, name);
                }
            }
            else
            {
                prefabAsset = String.Format("{0}/Prefabs/{1}.prefab",
this.assetPathToTiled2UnityRoot, name);
            }

            return prefabAsset;
        }
        public void Dispose()
        {
        }
    }
}
```

Appendix 4. CameraController.cs

```
using System.Collections;
using UnityEngine;

public class CameraController : MonoBehaviour
{
   public GameObject followTarget;
   private Vector3 targetPos;
   public float moveSpeed;

   private static bool cameraExists;

   // Use this for initialization
   void Start()
   {

      if (!cameraExists)
      {
         cameraExists = true;
         DontDestroyOnLoad(transform.gameObject);
      }
      else
      {
         Destroy(gameObject);
      }
   }

   // Update is called once per frame
   void Update()
   {
      targetPos = new Vector3(followTarget.transform.position.x,
followTarget.transform.position.y, transform.position.z);
      transform.position = Vector3.Lerp(transform.position, targetPos,
moveSpeed * Time.deltaTime);

   }
}
```

```
#if UNITY_4_0 || UNITY_4_0_1 || UNITY_4_2 || UNITY_4_3 || UNITY_4_5 ||
UNITY_4_6 || UNITY_4_7 || UNITY_5_0
#undef T2U_USE_ASSERTIONS
#else
// Assertion library introduced with Unity 5.1
#define T2U_USE_ASSERTIONS
#endif
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using UnityEngine;

#if T2U_USE_ASSERTIONS
using UnityEngine.Assertions;
#endif

namespace Tiled2Unity
{
    public class TileAnimator : MonoBehaviour
    {
        public float StartTime = -1;
        public float Duration = -1;
        public float TotalAnimationTime = -1;

        private float timer = 0;

        private MeshRenderer meshRenderer = null;

        private void Awake()
        {
            this.meshRenderer = this.GetComponent<MeshRenderer>();
        }

        private void Start()
        {
#if T2U_USE_ASSERTIONS
            Assert.IsTrue(this.StartTime >= 0, "StartTime cannot be negative");
            Assert.IsTrue(this.Duration > 0, "Duration must be positive and non-
zero.");
            Assert.IsTrue(this.TotalAnimationTime > 0, "Total time of animation
must be positive non-zero");
#endif
            this.timer = 0.0f;
        }

        private void Update()
        {
            this.timer += Time.deltaTime;

            // Roll around the time if needed
            while (this.timer > this.TotalAnimationTime)
            {
                this.timer -= this.TotalAnimationTime;
            }

            this.meshRenderer.enabled = timer >= this.StartTime && timer <
(this.StartTime + this.Duration);
        }

    }
}
```