VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Xiao Xuejian

DEVELOPING A TURN-BASED STRATEGY GAME ON UNITY ENGINE

Technology and Communication
2021

ACKNOWLEDGEMENT

I am grateful to my family for their support during my studies, and I thank all my tutors and other students who guided and helped me throughout my studies.
Special thanks to Dr. Ghodrat Moghadampour for his time and patience and for helping me.

In addition to my tutors, I would like to thank Vassar Summer University, the Campus Online website, for organizing many open courses, including the game engine course and programming course that I needed to complete my project.

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Degree Program in Information Technology

ABSTRACT

Author	Xiao Xuejian
Title	DEVELOPING A TURN-BASED STRATEGY GAME ON UNITY ENGINE
Year	2021
Language	English
Pages	51
Name of Supervisor	Ghodrat Moghadampour

The objective of this thesis was to develop a turn-based game for personal computers using the Unity game engine with the C# programming language. The game offers simple strategic gameplay where the player acts in the role of commander and controls different characters to complete the designed tasks. The Unity game engine was chosen because it offers cross-platform development capabilities, a wealth of third-party development resources, and is also free to users learning how to develop games.

First, a graphical user interface including three different game scenes was designed. Secondly, the game provides three difficult missions for players to choose from, the higher the difficulty means the more challenging it to complete the task. The difficulty of the task means the amount of reward. Then, game data files were designed, written, and modified in JSON (JavaScript Object Notation). Those files managed games and dialogues content.

The game can be further developed into a fully professional game, the content of the game could include more different enemy types and game maps could be added, more mission types and challengeable objectives could be designed, the equipment system could provide more equipment types and more equipment effects, and at the same time, the base construction content could be further developed.

Keywords Unity game engine, Turn-based strategy game, C# language

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF CODE SNIPPETS

1 INTRODUCTION

As the information age, or computer age, turns from science fiction to reality, video games have also emerged as a new type of game as a product of the information age almost at the same time, and it has been playing an increasingly important role in human life since its birth in 1948. /1/

Normally, people intricately linked the development of the video game industry with the development of information technology. In this thesis, the purpose is to observe and analyze the current game industry from a specific perspective, that is, the impact of the development of mobile communication technology on the game industry.

The development of the game market worldwide is almost driven by the form of the game, but in China, the main condition for the development of games is the development of communication technology.

In 1998, China's game market only had a scale of 1 million US dollars (calculated at the current exchange rate of that year), of which domestic games were only 100,000 US dollars. After 20 years, the revenue of the game market has increased by 2680 times which is $ 34 billion (calculated at the current exchange rate). Although, as seen in Figure 1 below, the game market scale expands rapidly, it cannot respond well to the development of China's actual game industry because of people’s income growth. So, more data and figures can help analyze what happened. /2/

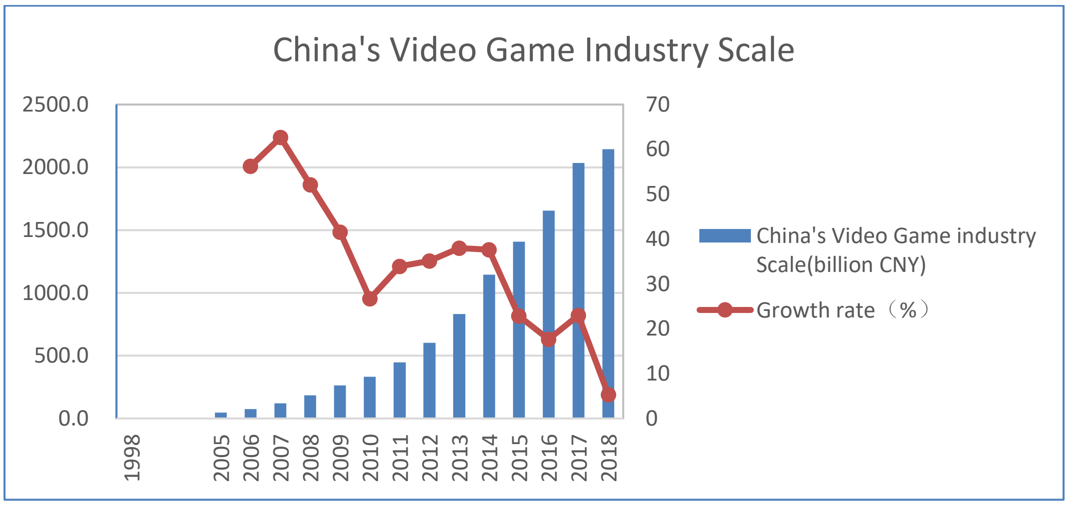


Figure 1. China's Video Game Industry Scale /2/

The game engine technologies had also entered a mature stage with the development of the game industry. Developers can focus more on game design instead of repetitive workload across different platforms and reduce the difficulty of game development by using those public game engines. It also brought a possibility to develop games quickly and easily for multiple platforms.

A strategy video game is a video game that focuses on skillful thinking and planning to achieve victory. /3/ In the past 30 years, games are subdivided into many different types. Strategy video games, which evolved from table games, also are subdivided into two different game types: real-time and turn-based. Real-time strategy games indicate all the players’ actions in the game in continuous time. It includes plenty of popular games such as Starcraft series, Warcraft series, and DOTA series. TBS, also known as a turn-based strategy that is distinguished from real-time strategy, asks players to take turns as a sequence when playing. Representative games include Sid Meier's Civilization series, Heroes of Might and Magic series, and so on.

This thesis presents the learning of the unity3D engine and other software required to design and implement digital games, then design and implement a

2D/3D turn-based game on the PC platform for the Windows Operation System as the thesis project for the Information Technology department at Vaasan ammttikorkeakoulu, University of Applied Sciences. The topic of this thesis came from my interest in the games, my observation and understanding of the current state and development of the game industry, and the attempt, and a summary of game development.

1.1 Game Design and Objectives

Many people follow different ways to guide the implementation process from the idea to the actual work. For game design, the design method guides the designer to analyze the goals and requirements to achieve the project, and then adjust to realization. Finally, depending on the result the implementation is refined. Game developers, game designers, researchers, and players exchange opinions through this set of frameworks so that everyone can understand each other. Not only can it help game designers with game design and analysis, but it can also guide players to have a better experience of the game from a more professional perspective. /4/

Here one popular design template to design the game is introduced.

1.1.1 Game Planning Template

There are many types of game-planning templates. A template suitable for developers without much development experience was mainly used in the project.

The game as follows:

1) **One Sentence**: A space science-fiction style strategy game

2) **Keywords:** Turn-based, strategy element, Space base development

3) **Introduction:**

   A space science-fiction style strategy game that players can develop their space base, train team roles, challenge missions of various difficulty.

4) **Features**:

   Improvements to traditional turn-based strategies.

   Space science fiction art style.

5) **Game experience:** An appropriate difficult strategy and management game with space base development elements.

6) **Art style:** Space-science fiction art style.

7) **Music sound effects**: Due to independent development, sound effects are not a priority.

1.1.2 Objectives

The game will be several stages to complete, in this thesis, only the content of the first stage will be implemented.

Stage 1: Thesis game project

In this state, the main task is to become familiar with various game development tools, for example, to learn unity3D engine by reading the documents and tutorials includes the user manual, script API and video tutorials. Three basic game scenes should be finished, it includes the main menu scene that supports functions players start the game and change settings of the game, the game scene to manage the team and choose the combat missions, and the battle scene that contains core elements of the game. It took several months.

The map of game design for the first stage is shown in Figure 2.

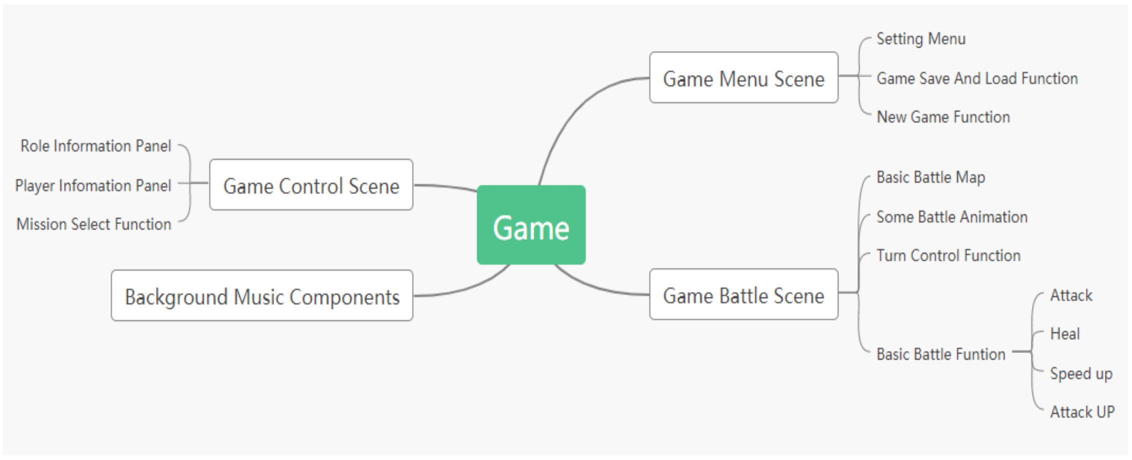


Figure 2. Game Design Plan

Stage 2: Game Content Complete

In this stage, the main tasks focus on the implementation of all base construction and game roles operations-related content and improve the design of game combat features such as game role skills, tactics for combat, and game mission map. This was planned to be completed in one month.

Stage 3: Reconstruction of the code structure

Because the previous stage was more about learning and understanding, there was no systematic program structure design. At this stage, all codes will be re-edited to follow a popular software framework such as the MVC framework or the MVVM framework for Unity3D. Planned to be completed in one month.

Stage 4: Completion for artistic style and sound effects

As a person who lacks the knowledge and experience of art and music, completing the game art style was envisioned a difficult challenge. But as the most important feature of the game, art style determines the most gaming experience of players. The learning of how to use 3D design software will be continued, such as blender, to design and fulfill all the game models, animations, effects, shaders, game background scenes. Three months of study and development is planned.

1.2 Motivations and Project Topic Description

The motivation for the thesis arose from the author's desire to seek a game designer position in the gaming industry. To combine previous studies gained from college life, get familiar with the real game development process, and get individual game design and implantation a complete game as an employment advantage, developing a game on a popular development environment in the industry was selected as the topic of the project and the thesis.

This thesis focuses on a typical type which is the turn-based strategy game. As mentioned above, the project aimed to design and implementation of a TBS game face to a personal computer platform base on the Unity game engine.

2 RELEVANT TECHNOLOGIES

This chapter describes all software which were used during the development process. Some technologies used during the development of the thesis are also listed here.

2.1 Game Engine

The Unity game engine provides supports to produce both 2D and 3D games and aid in porting them to multiple platforms. It mainly offers a C# programming language developing environment and much technical documentation. Almost all engine features are free to all the personal users to develop games for non-commercial use and three paid licensing options for commercial use which also offers additional functions. /6/

2.1.1 Unity Game development Interface

An example picture of Unity game development interface is shown in Figure 3.

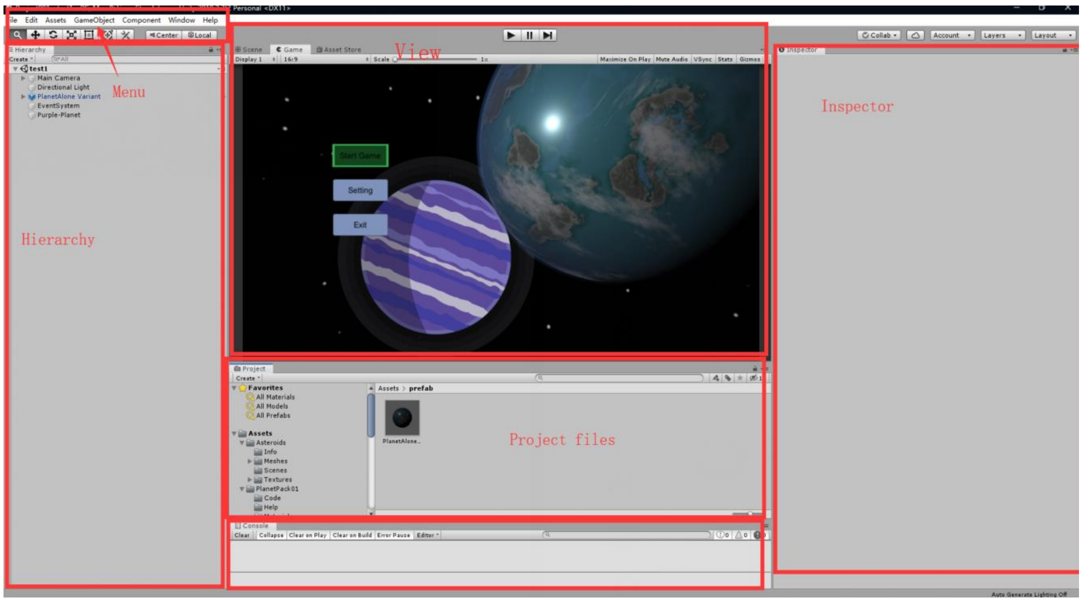


Figure 3. Game development Interface

The Unity interface offers many windows which act different functions, mainly include:

a) Project files browser is a file system that shows game files (game assets) and location on your computer.

b) Hierarchy is a menu that the user can manage. The user’s game element includes UI elements, game objects, and so on.

c) Menu and Toolbar provide the Unity menu which helps the user to control unity and your game project.

d) View windows contain Scene View, Game View, and Asset store view.

Scene View graphically shows the user’s game elements and objects. This View presents the project developing process and helps users to edit the game directly and quickly.

Game View displays the current project’s real-time render screen sense without building the project.

Unity offers an Asset Store that the user can purchase and get game developing assets including customized unity plugin and tools, 2D/3D art resources, audios, and so on.

e) Inspector, the Inspector is used to view and edit the properties and settings of almost everything in the Unity Editor.

f) Console, errors, and debug information will be shown here.

2.2 Visual Studio

Unity support Visual Studio instead of directly provide programming windows or functions. Users can also use another IDE to edit their script files.

With the development of unity, it also offers some no programming functions, but scripts work still is the main content of the process of implements of users' project. In Unity, the user can access most functions by scripts, they are different from the general interface development environment, but Unity support much more powerful graphics functions.

New C# files functioning as script files should be created under the Assets file dictionary. Usually, other assets such as art resources, plugin, and tools should import here also, and user manage their files in a different subdirectory.

2.3 Blender and Photoshop

Blender is a free and open-source 3D computer graphics design toolset software. Because Unity does not support professional graphics design functions. Blender was used to design some game model objects and animations during the implementation of the project. Blender's mainly support 3D model design, texture edit, shader design, and animation. It becomes more and more popular because quite a lot of companies and organizations have started to support the development of Blender, in the game industry. It will become one of the universal designs and development tools. /7/

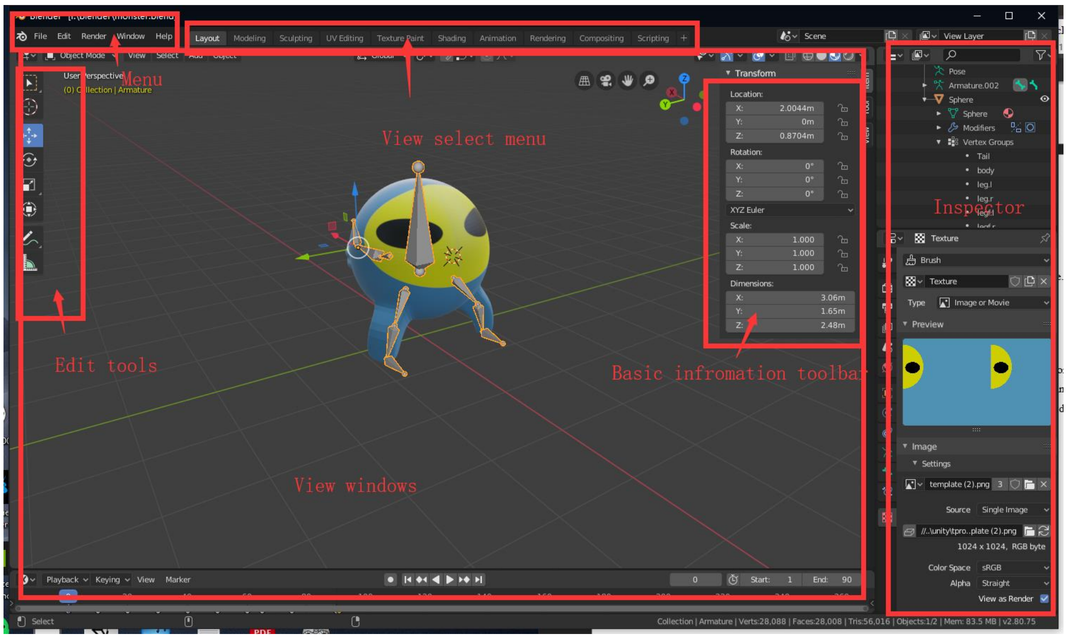


Figure 4. Blender Interface

Photoshop is a picture editor software. The user can apply almost any changes to digital graphics, it is mainly used to make textures and UI materials during the project implements.

2.4 JSON

JavaScript Object Notation, usually abbreviated as JSON, is a lightweight data-interchange format which easier for people to read and program and is used to install or transfer data objects consisting of attribute values or sequential values. It grew out based on the JavaScript Programming Language in 2001 and became popular from 2005 until now. /8/

2.5

LitJSON is a small open source .Net library that is used to handle and operate JSON files. It is written in the C# programming language which helps programmers to use JSON easier and save a lot of time at the same time. /9/

3 APPLICATION DESCRIPTION

The objective is to develop a turn-based strategy game. Turn-based means that the game will have a clear sequence for the player to complete actions on the game objects, while strategic means that the game will offer different actions to achieve different effects, and the player will need to think and choose to get different game results. The background of the game will be space elements as the theme. Players complete missions, get rewards, use the rewarded resources to strengthen themselves, and experience the game content through this cycle.

3.1 Quality Function Deployment

As the objectives mentioned before, the development of this thesis project will be finished in four stages. Here is the description following the rules of quality function deployment. The first stages can be also considered as three categories: features must be finished, features expected to be finished, and the optional or attractive features.

3.1.1 Must-Have Requirements

Because it is a game project, the basic functions of a playable game must be done. The functions included are listed in Table 1.

Table 1. The list of must have requirements.

Game Menus	Different Menus needed to start a game and access other game control functions.
Game objects and Game map	As a turn-based strategy game, game roles which controlled by the player, enemies wait to be challenged, and the basic

	playfield that the player can play the combat should be built.
Attributes of roles and enemies	As a strategy element, the differentiated character attributes should have different effects on the game
Skills of roles and enemies	Basic game skills or game strategy options during the combat should be finished.
Turn Functions	Turn-based system to control the battle process of the game.
Rewards	A basic rewards system, usually money or point in the game.

3.1.2 Should-Be Requirements

After the most basic game functions are completed, more designs are still needed to make the game experience more complete.

Table 2. Should Be Done

Background Music	Simple Background music will make the game less boring
Info Panels	Info panels that show game status; include player and roles information.
Difficulty selection	The simple difficulty selection function replaces the game map to guide the game flow
Setting Menu	A setting menu for player to adjust various settings of the game
Save and Load Function	Used to save game data and continue previous games

3.1.3 Nice-To-Have Requirements

Completing more interesting and attractive optional features will make the game more complete.

Table 3. Optional Part

Game items and Inventory	Including a variety of game items will make the game more strategic, and at the same time, the playability will be greatly improved. The Inventory function is designed to manage these items
Game story and game map	The game story and game map functions are used to guide players to experience the game content
More optional characters and rich enemy types	Used to fill and complete the game content
Character development	The character development system is used to train the player's roles

3.2 Use-case Diagram

This use-case diagram (Figure 5) indicates how the player, or the user controls the game system. After starting the game, a player can start a new game, or load the previous save files for the game. The player can adjust the settings of the game. The player can access the game control function by choosing different game save files. Then the player can start game combat by choosing a different difficult game level to win the rewards.

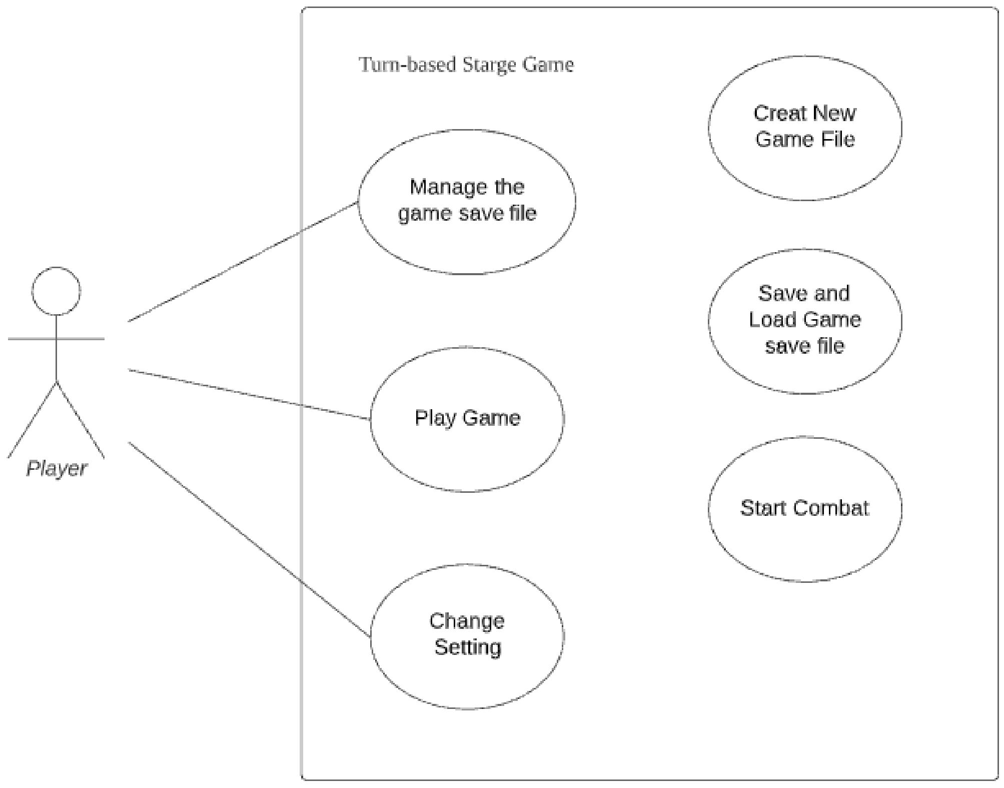


Figure 5. A use case diagram for player

Use cases diagrams below indicate the player's activities in detail.

3.3 Sequence Diagram

The sequence diagram In Figure 6 presents how the player plays the game and manages the game save files and game setting.

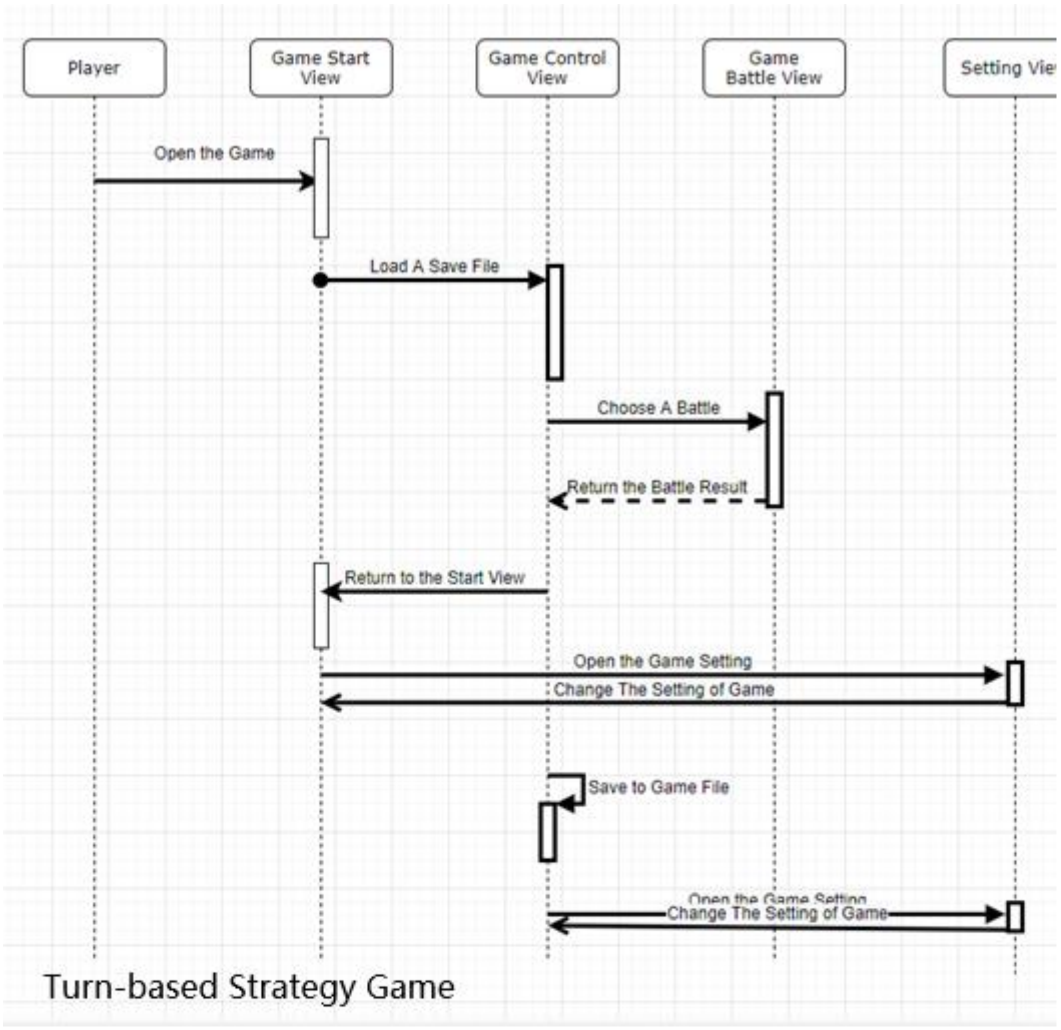


Figure 6. Sequence diagram

The sequence diagram description step-by-step:

1) Players opens the game, and the game starts a view.

2) Players choose to start/load a game, then change to the game control view.

3) Players choose challenge level and star game combat, the game switches to the game battle View.

4) Win or Lose the combat, battle results will return to the game control view.

5) Players can return to the game start view from the game control view.

6) Players can access the setting functions at the game start/control view.

7) Players can save the game to save files at the game control view.

3.4 Class Diagram

Because the project is developed with a Unity engine, a common class provided by the official library, the Monobehaviours, should be inherited by all the class which running with the start and running of the game engine.

The class diagram in Figure 7 describes the class relationship of the game start view and game control view. The class diagram in Figure 8 describes the class relationship of the game combat view.

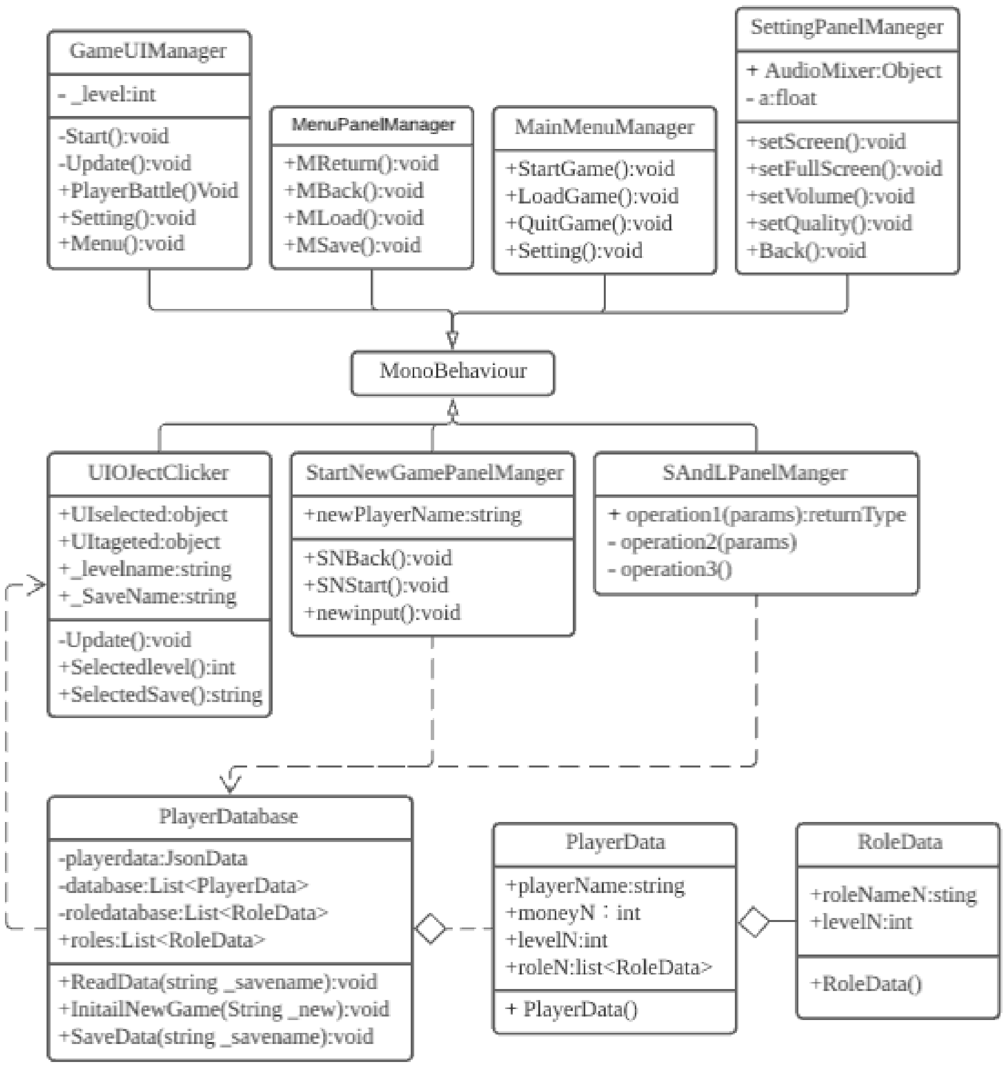


Figure 7. Game Start / Control Class diagram

Figure 7 shows the UI Object clicker which checks the mouse click game object and returns the information of the object, saves, and loads the panel manager, starts the new game panel manager, menu panel manager, game UI manager, main menu manager, setting panel manager. Those manager classes control almost all UI components and their functions through the interfaces IButton, IDropDown, ISlider, and IToggle, then implement by different function classes. The PlayerDatabase class manages the JSON save files.

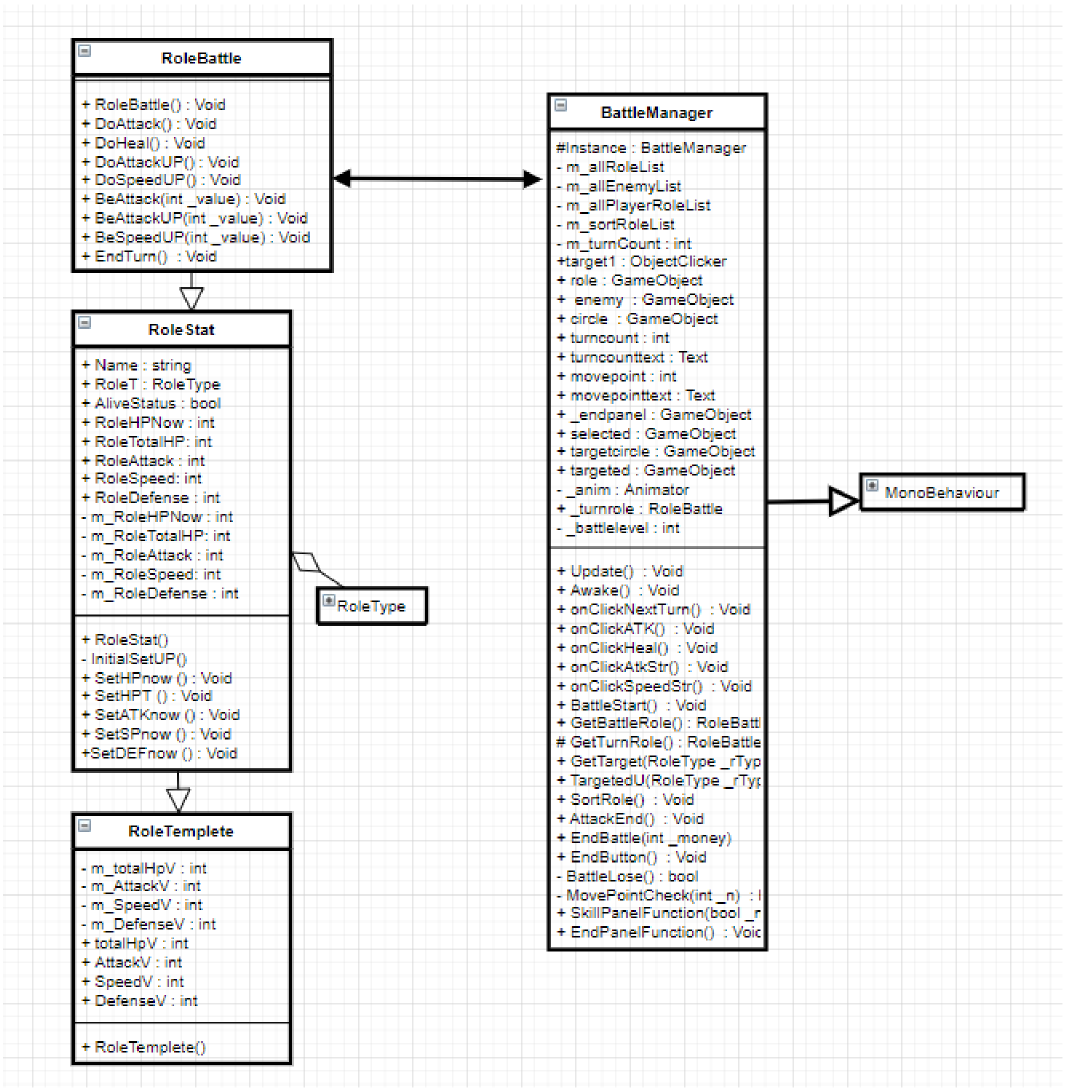


Figure 8. Game Combat diagram

As shown Figure 8 the battle manager class control all the functions for the game combat view, it manages the turns and sorts the player roles and enemies' turn order. The UIObjectClicker function that helps the player select the target by mouse click and checks the result of the combat after the end of each turn. RoleBattle defines the player's roles and enemies fight in combat, the skills, and functions implementation.

4 DATABASE

This game project uses the JSON files as the database. Those files are used to store the main game data, such as player archives, dialogue information, and mission information.

4.1 Design of the Database

The game is designed to use JSON to save the game. A simple save file structure shows below:

```
{
"playerName": "123",
"moneyN": 1003000,
"levelN": 1,
"roleN": [
{
"roleNameN": "New1",
"levelN": 1
},
{
"roleNameN": "New2",
"levelN": 1
}
],
"ISlot": [
{
"inventorySlot": 0,
"itemIDS": -1
},
{
"inventorySlot": 1,
"itemIDS": 0
}
```

Code Snippet 1. JSON Save File structure

5 IMPLEMENTATION

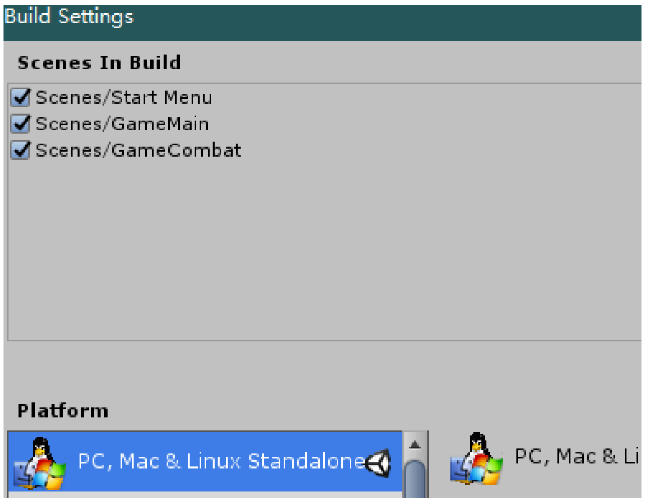
Because the aim is to develop independent games, assets provided by others was avoided during the development of this thesis project. In this chapter, the focus is on the implementation functions and features, and some learning results of Blender.

5.1 Structure of Project Game

After a new project was created, the usual developer will create a series of folders under the Project/Assets directory. All the game assets include scripts, scenes, sprites, textures, material maps, shader maps, font files, models, prefabs, animations, and so on, should be managed storage by categories. A good file management habit can greatly reduce the confusion of files.

5.1.1 Game Scene Manager

In the Unity engine, different scenes are managed by the Build Settings Panel. All the scenes can be added here to build a runnable game, and by defining the order numbers to achieve a different scene switching function. In this game project, three scenes have been created for different functions.



Scenes/Start Menu
Scenes/GameMain
Scenes/GameCombat

Figure 9. Build Settings Panel and Start Menu Scene

5.1.2 Start Menu Scene

In this scene, the game offers an interface that includes different UI Components and supports basic management to the settings, starts a new game, or quit the game by corresponding button functions.

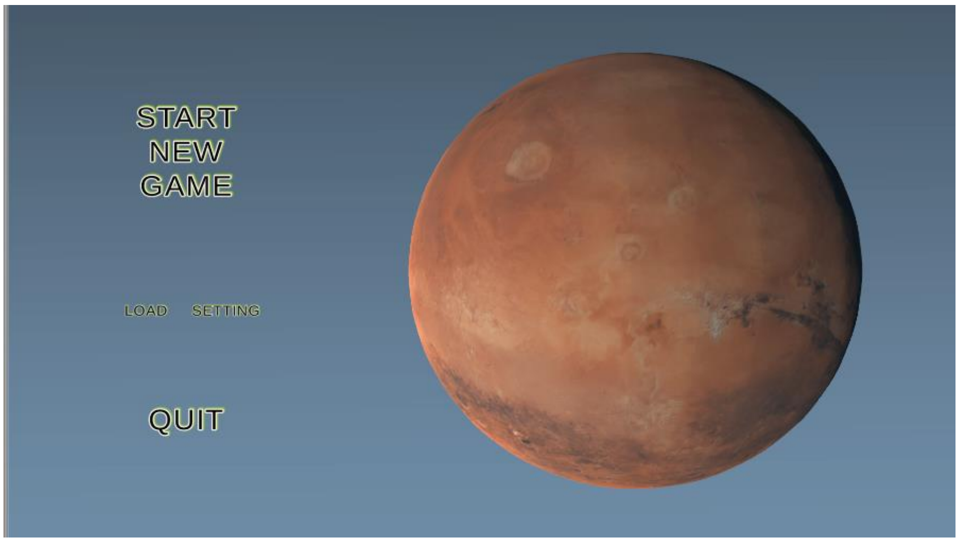


Figure 10. Start Menu Scene

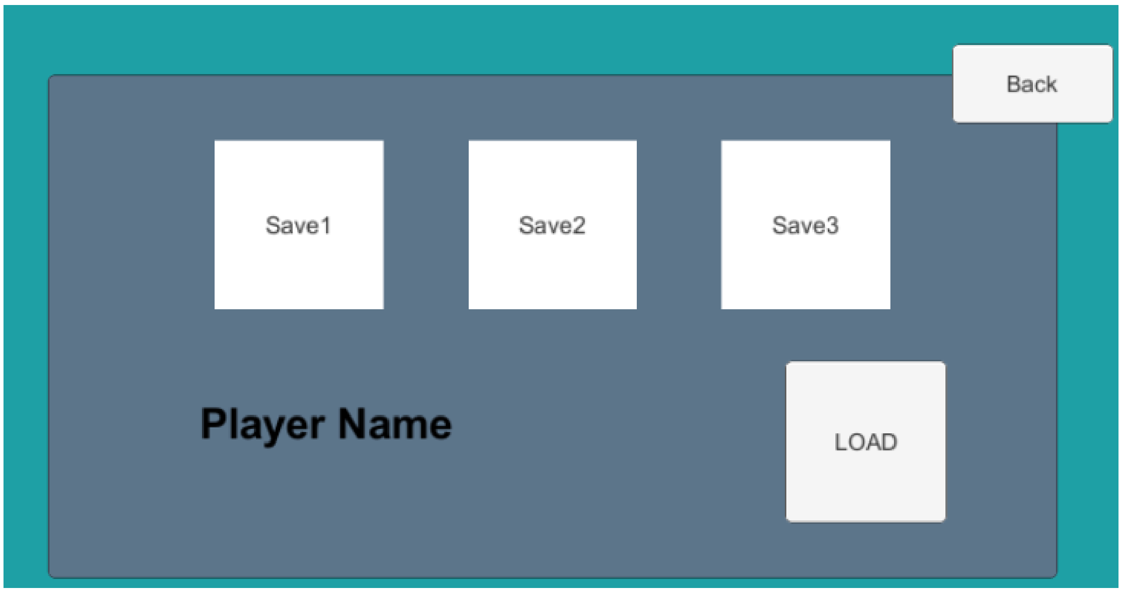


Figure 11. Load Game Menu View

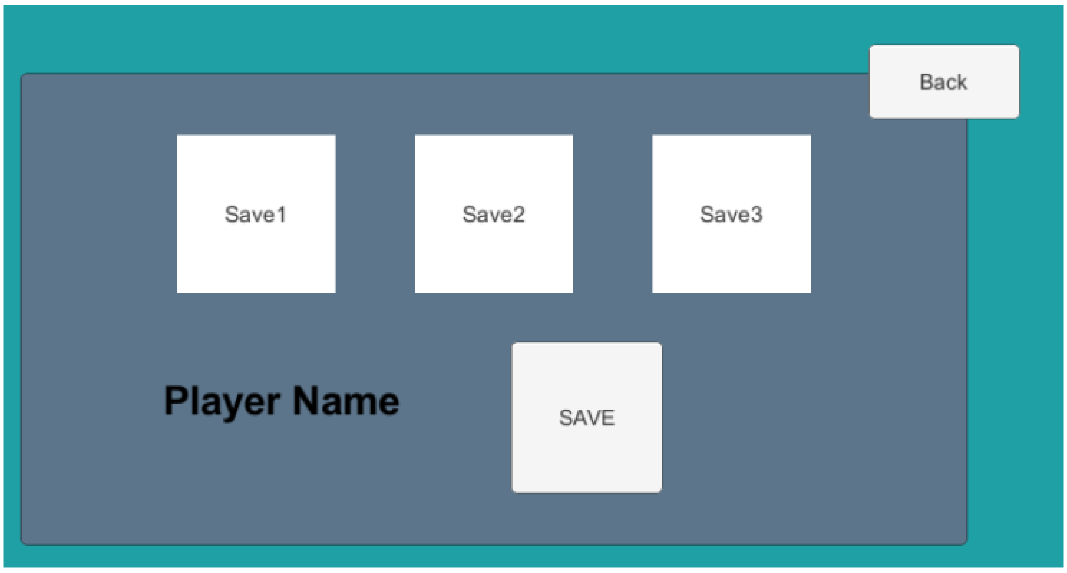
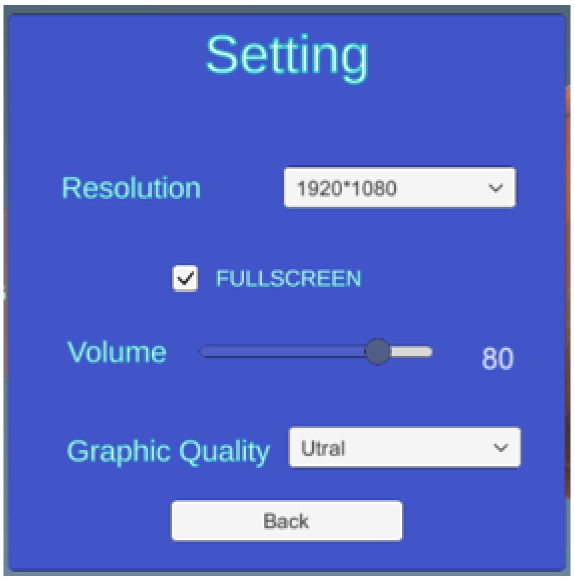


Figure 12. Save Game Menu View


Setting

Resolution: 1920*1080

✓ FULLSCREEN

Volume: 80

Graphic Quality: Utral

Figure 13. Setting Panel View

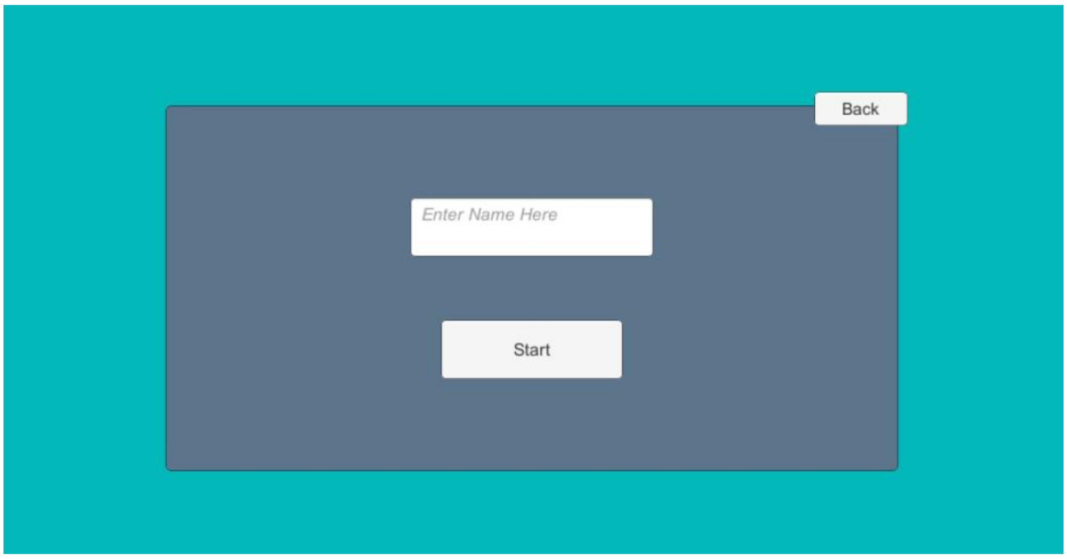


Figure 14. Start New Game View

5.1.3 Game Main Scene

This scene (Figure 15) offers an interface that includes different UI Components and supports different challenge level selections to the combat. The player can also adjust the setting of the game or return to the start main menu Scene.

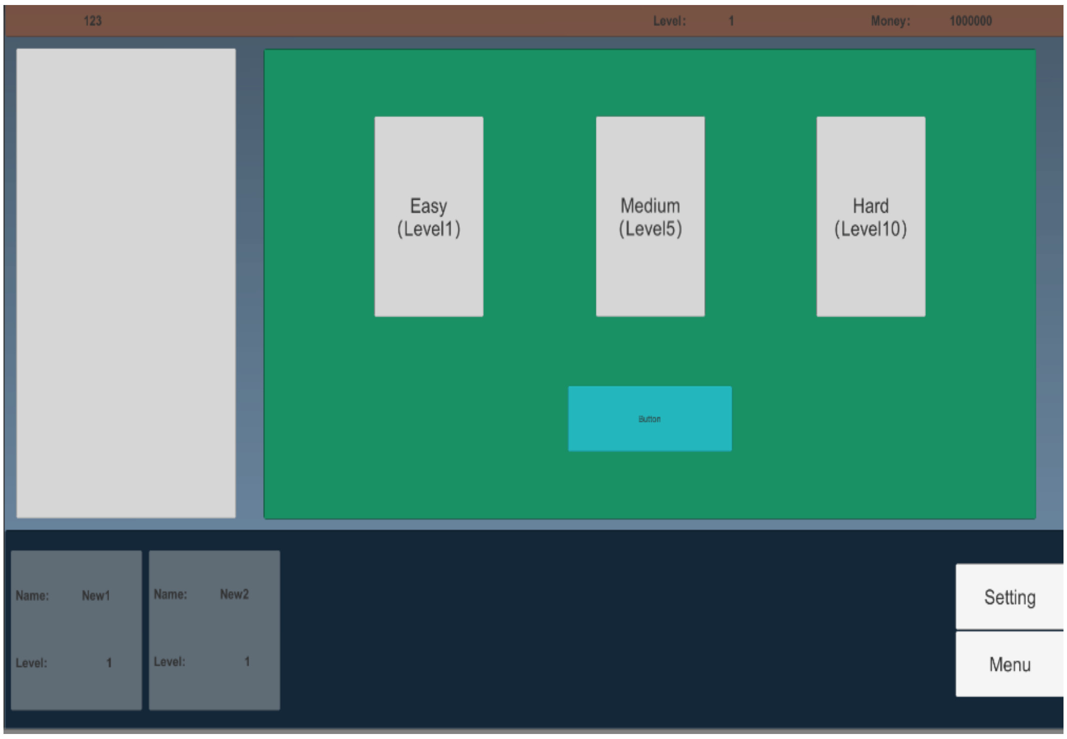


Figure 15. Game Main Scene

There are three challenge level for the player to choose. Choosing different difficulty levels will weaken or strengthen the enemy, at the same time, the reward will be decreased or increased.

5.1.4 Combat Scene

The Combat Scene is the most important scene in this game object. The player can control the game roles to fight with the enemies. Different skills have different functions. After the game combat loses or wins, a result view will show to the player, then, the player can return to the game main scene.

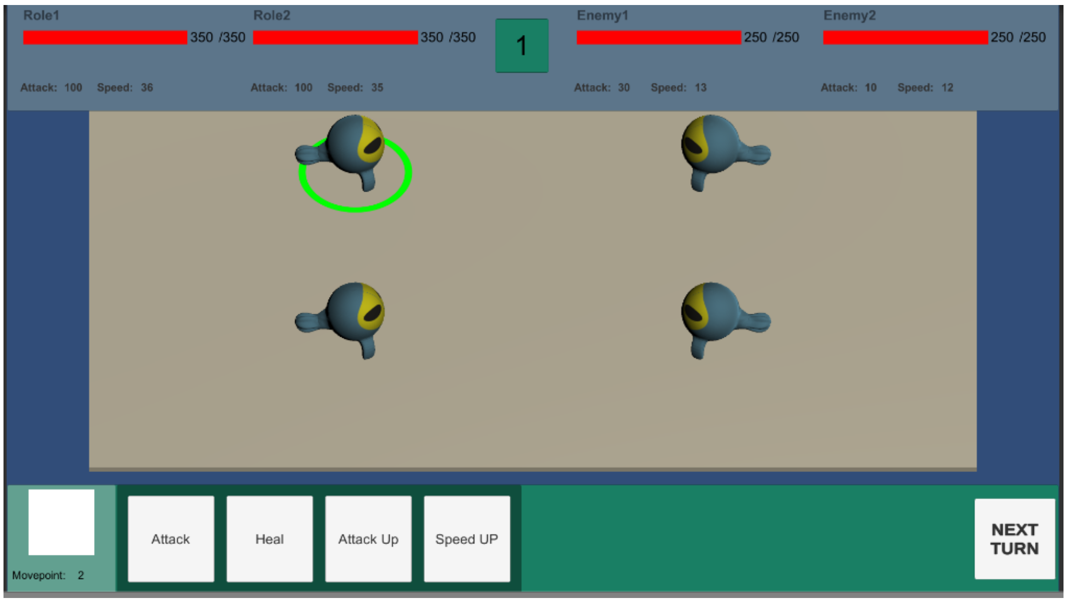


Figure 16. Start Menu Scene

Once the turn starts, a green circle will indicate the target follows the turn order. The player can select a target as the skill-affected object. Then the move point will be decreased by different skills player release. Once it equal 0, the player cannot do anything more but click the next turn button. Then the next turn hold target which will starts the turn of it.

The HP points show the roles and enemies' statuses. If it equals 0, the statuses will set to die, at the end of each turn, the game will check the battle result.

In the view shown in Figure 17, the players win/lose the combat and get their rewards. Then play will return to the game main scene by clicking the Back to Game button.

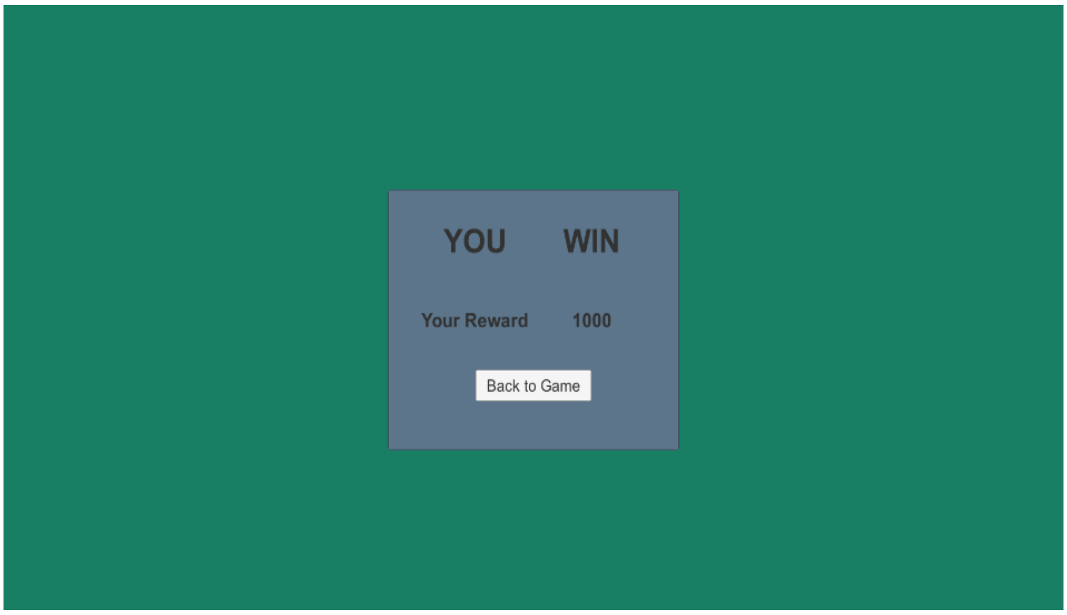


Figure 17. Result View

On this view, the players win/lose the combat and get their rewards. Then play will return to the game main scene by clicking the Back to Game button.

5.2 Game Model and Animation

5.2.1 Game Model

A game model is a set of game assets that usually includes game characters, items, environmental elements such as sky, stone, trees, and so on. The game developer using those models to create the scene and, interactive elements for the game.

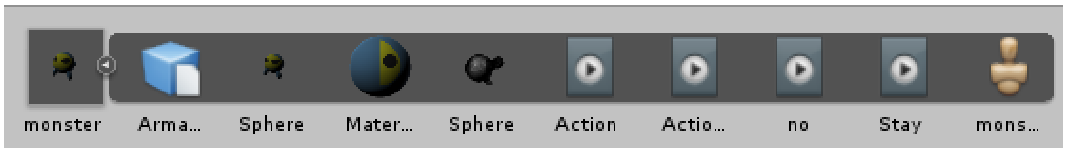


Figure 18. Game Model with Armature and Animation Slice

5.2.2 Texture

Textures are image or movie files. The correspondence with the appearance of the model or GameObject in various ways.

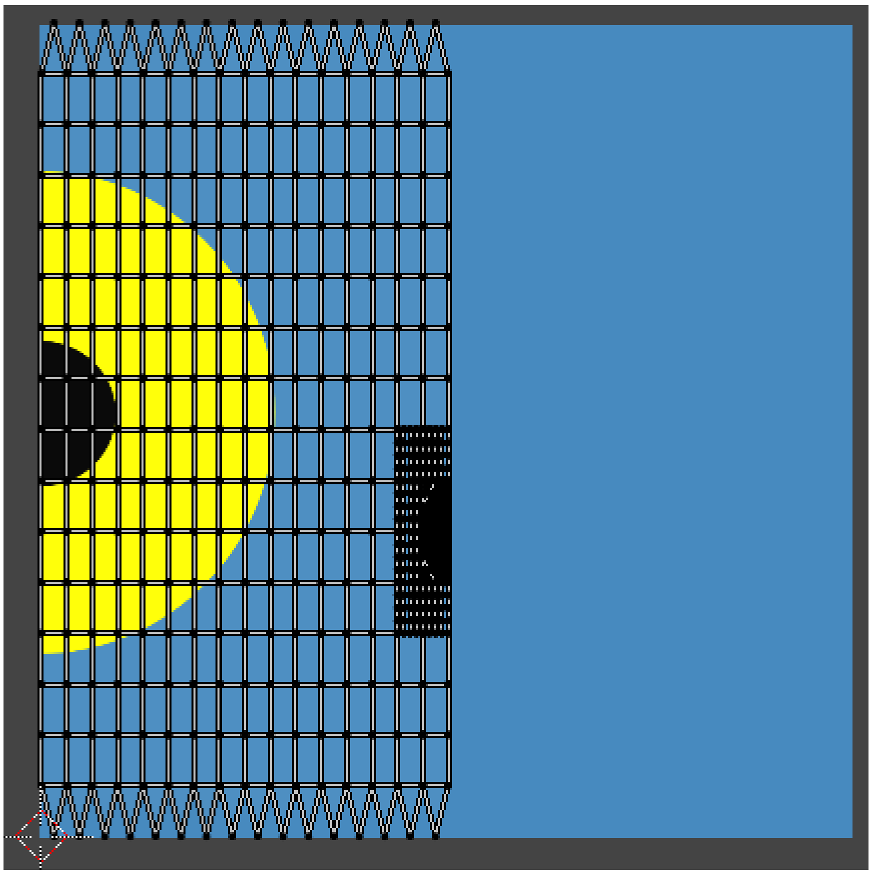


Figure 19. Texture with UV map

5.2.3 Animation

Unity uses an animation controller to control the animations for a specific character or GameObject. The animation clips of a GameObject can be switched by an animation controller map; the developer can access and control them by program or a visual programming language with unity.

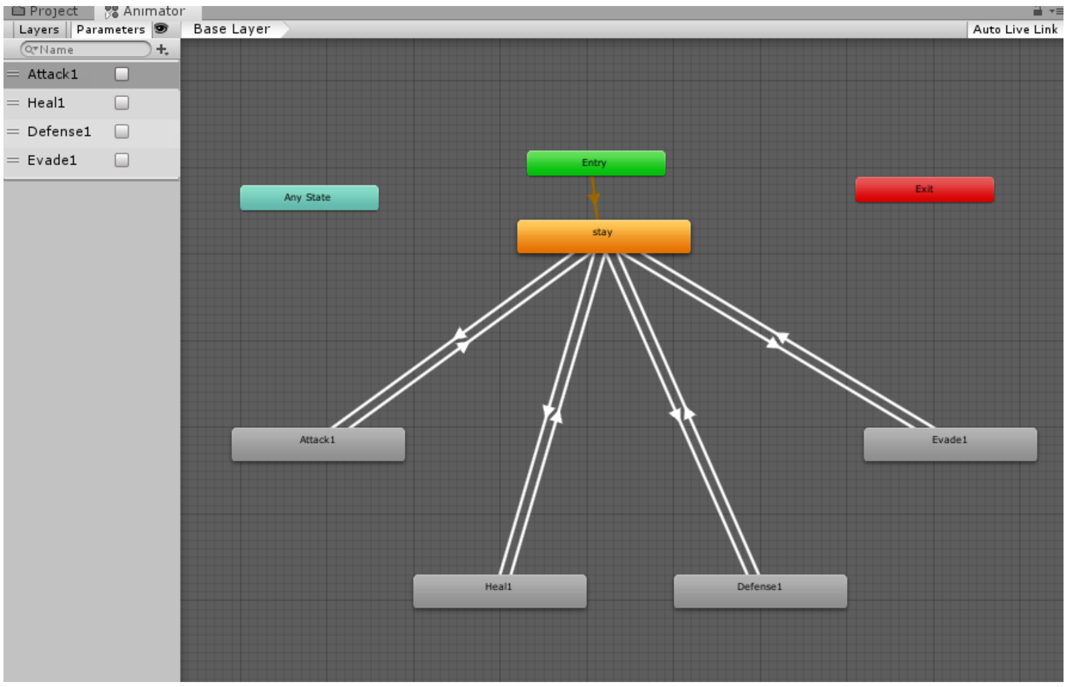


Figure 20. Animator controller map

5.2.4 Prefab

Unity’s Prefab supports a function that the developer can store a set of GameObject as a generic package. When a prefab is created, that means the next time if it is directly used to copy a set of GameObject as one GameObject then much fewer components need to be modified instead of than adding those GameObject again and again. But the original prefab cannot be changed unless to there is a need to change all the GameObject created by this prefab.

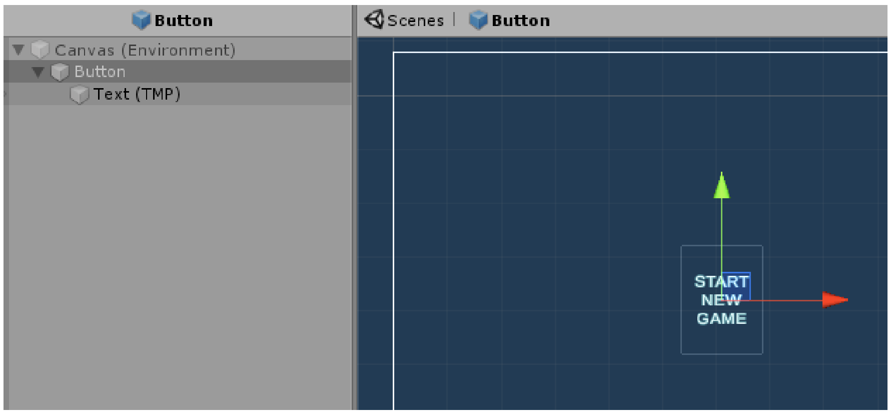
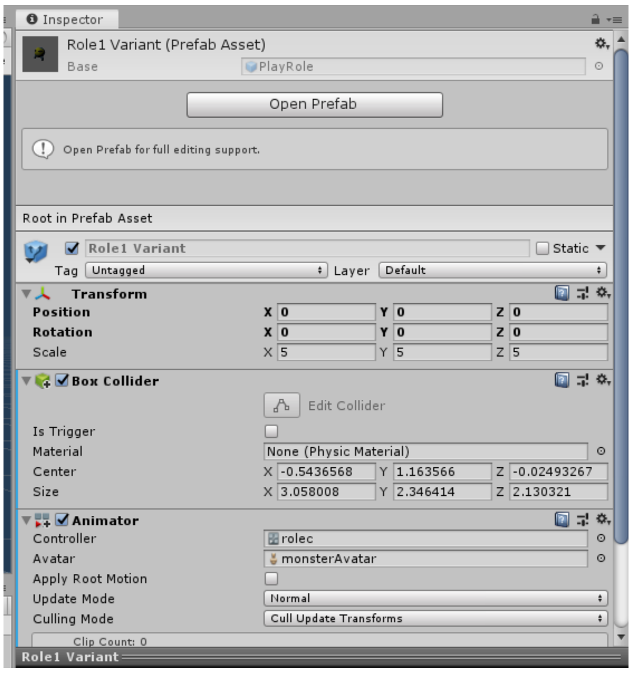


Figure 21. A UI prefab example

The system allows the creation, configuration, and storing of a GameObject complete with all its components, property values, and child GameObject. As a reusable Asset, the Prefab Asset acts as a template from which new Prefab instances in the Scene can be created.



```
Inspector
Role1 Variant (Prefab Asset)
Base                PlayRole
Open Prefab
Open Prefab for full editing support.
Root in Prefab Asset
Role1 Variant                          Static
Tag Untagged                  Layer Default
Transform
Position            X 0          Y 0          Z 0
Rotation            X 0          Y 0          Z 0
Scale               X 5          Y 5          Z 5
Box Collider
                    Edit Collider
Is Trigger
Material            None (Physic Material)
Center              X -0.5436568 Y 1.163566   Z -0.02493267
Size                X 3.058008   Y 2.346414   Z 2.130321
Animator
Controller          rolec
Avatar              monsterAvatar
Apply Root Motion
Update Mode         Normal
Culling Mode        Cull Update Transforms
Clip Count: 0
Role1 Variant
```

Figure 22. A 3D model Prefab

5.3 Script Implementation

This chapter indicates the implementation of the game. Different script resources are introduced and described. Because the Unity engine offers many libraries, only necessary functions or classes from the unity libraries will be included.

5.3.1 The Interface of UI Component

This is the script of the interface for most UI component, it includes buttons, sliders, toggles and dropdown boxes.

```
namespace GameUI {
  interface IButton
  {
    void ClickFunction();
  }
  interface IButtons
  {
    void ClickFunction(string _n);
  }
  interface ISlider
  {
    float SliderFunction(float n);
  }
  interface IToggle
  {
    void Statechange();
  }
  interface IDropdown
  {
    void DropDownFunction();
  }
  interface IDialogue
  {
    void DialogueBox();
  }}
```

Code Snippet 2. The Interface of UI Script

The interface (Code Snippet 2) defines the different methods to be implemented.

5.3.2 UI Manager

Figure 23 shows one example of that UI manager which indicates how the UI component functions managed by different scripts. Because Unity supports visual programming, the developers do not need to fully implement all functions in code. the corresponding method can be edited in the script, and the UI components modified such as buttons, on the Unity developing interface.

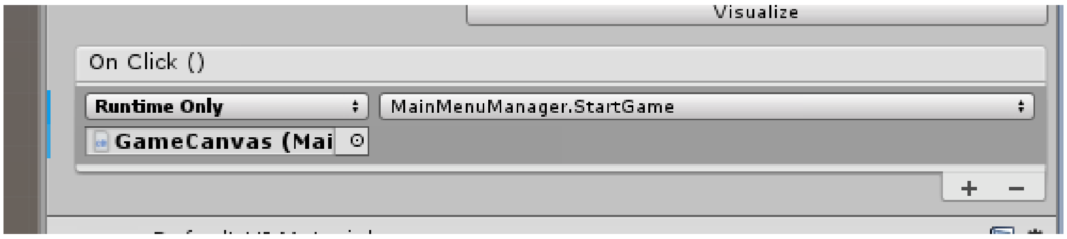


Figure 23. Example of a visual programming

```
public class MainMenuManager : MonoBehaviour
{
   public void StartGame()
   {
      IButton buttonfunction = new BStart();
      buttonfunction.ClickFunction();
   }
   public void QuitGame()
   {
      IButton buttonfunction = new BQuit();
      buttonfunction.ClickFunction();
   }
   public void LoadGame()
   {
      IButton buttonfunction = new BLoadPanel();
      buttonfunction.ClickFunction()}
   public void Setting()
   {
      IButton buttonfunction = new BSetting();
      buttonfunction.ClickFunction()}}
```

Code Snippet 3. Main Menu Manager Scrip

The required class is created through the interface and the corresponding function is called.

5.3.3 UI Component Function

Code snippet 4 is an example how to implement the UI Component Function interface.

```
namespace GameUI {
  public class TFullScreen : IToggle
  {
    public void Statechange() {
      var a = GameObject.Find("/Main
Camera/GameCanvas/SettingPanel/FSToggle").GetComponent<Toggle>();
        Debug.Log(a.isOn + "123");
        Screen.fullScreen = a.isOn;
        Debug.Log(Screen.fullScreen);
    }}}
```

Code Snippet 4. Method to Change Fullscreen Status.

A new GameObject is created and corresponded to the object in the corresponding directory and call the relevant functions through this GameObject.

5.3.4 The Setting of the Game

```
public class SettingPanelManager : MonoBehaviour
{
  //here defines the Audio Mixer control and a float value control the volume
  public AudioMixer audioMixer;
  float a = 0;
  //here is the function to set the screen resolution
  public void setScreen()
  {
    IDropdown Dropdownfs = new RDrop();
    Dropdownfs.DropDownFunction();
  }
  //here is the function to change the fullscreen state
  public void SetFullscreen()
  {
    IToggle FScreen = new TFullScreen();
    FScreen.Statechange();
  }
  //here is the function to change the BGM volume
  public void SetVolume()
  {
    ISlider Volumefs = new SVolume();
    a = Volumefs.SliderFunction(a);
    //Debug.Log(a);
    audioMixer.SetFloat("MasterVolume", a / 2 - 40);
  }
  //here is the function to change the Graphic Quality
  public void SetQuality()
  {
    IDropdown Dropdownfs = new QDrop();
    Dropdownfs.DropDownFunction();
  }
  //here is the Close the setting panel
  public void Back()
  {
    IButton buttonfunction = new BSetting();
    buttonfunction.ClickFunction();
  }}
```

Code Snippet 5. Setting Menu Manager Script

Different functional methods achieve different functions. This includes adjusting the sound level by sliding the slide component, choosing different screen options by selecting the drop-down box, choosing whether to go full screen by ticking the component and choosing different screen resolutions by selecting the drop-down box.

5.3.5 Combat System Initialize

When a player chooses to start a combat, the game will receive the level setting of the challenge, and then adjust the roles and enemies' initial settings which showed in Code Snippet 6.

```
RoleBattle _role1 = new RoleBattle(4,0,"Role1", RoleType.Player);
_role1.SetSPnow(1);

GameObject.Find("BattleUICanvas/InfoPanel/InfoPanelS/Role1/Slider").GetComponent<Slid
er>().maxValue = _role1.RoleHPNow;

GameObject.Find("BattleUICanvas/InfoPanel/InfoPanelS/Role1/Slider").GetComponent<Slid
er>().value = _role1.RoleHPNow;

GameObject.Find("BattleUICanvas/InfoPanel/InfoPanelS/Role1/HPTotal").GetComponent<Te
xt>().text = " /" + _role1.RoleHPNow.ToString();

GameObject.Find("BattleUICanvas/InfoPanel/InfoPanelS/Role1/HP").GetComponent<Text>()
.text =  _role1.RoleHPNow.ToString();

GameObject.Find("BattleUICanvas/InfoPanel/InfoPanelS/Role1/Attackv").GetComponent<Te
```

```
xt>().text = _role1.RoleAttack.ToString();

GameObject.Find("BattleUICanvas/InfoPanel/InfoPanelS/Role1/Speedv").GetComponent<Te
xt>().text = _role1.RoleSpeed.ToString();
```

Code Snippet 6. A game role set up example to initial the combat.

Then the game will create a visual GameObject from prefabs on the specific coordinate position. Then the name of those GameObjects are set to the right name.

```
PlayerRole1 = Instantiate(role, new Vector3(-300F, 0, 300F), Quaternion.Euler(0, 0,
0),GameObject.Find("Roles").transform);
PlayerRole1.name = "Role1";
PlayerRole2 = Instantiate(role, new Vector3(-300F, 0, -100F), Quaternion.Euler(0, 0, 0),
GameObject.Find("Roles").transform);
PlayerRole2.name = "Role2";
```

Code Snippet 7. Script to create the from prefab.

The figure below is the example script for basic methods to set all the attributes of the roles and enemies. The skill methods were also implemented through this class and those methods.

```
private void InitialSetUp(int _StatusLevel)
  {
    m_RoleTotalHP = TotalHpV[0]+ TotalHpV[1] * _StatusLevel;
    m_RoleHPNow = TotalHpV[0] + TotalHpV[1] * _StatusLevel;
    m_RoleAttack = AttackV[0]+AttackV[1] *_StatusLevel;
    m_RoleSpeed = SpeedV[0]+SpeedV[1] *_StatusLevel;
    m_RoleDefense = DefenseV[0]+DefenseV[1] *_StatusLevel;
  }

  public void SetHPnow(int _value)
  {
    m_RoleHPNow += _value;
  }

  public void SetHPZero()
  {
    m_RoleHPNow =0;
  }
```

```
public void SetHPFull()
{
   m_RoleHPNow = m_RoleTotalHP;
}

public void SetHPTL(int _value)
{
   m_RoleTotalHP += _value;
}
public void SetATKnow(int _value)
{
   m_RoleAttack += _value;
}
public void SetSPnow(int _value)
{
   m_RoleSpeed += _value;
}
public void SetDEFnow(int _value)
{
   m_RoleDefense += _value;
}
```

Code Snippet 8. Initial the basic characteristic of roles and enemies.

Obtain different values by assigning different values to different methods.

5.3.6 Combat Turn Control

After the game combat is initialized, the game will add those roles to a role list, add the enemies to an enemy list, and add them all to an all-roles list.

The role list and enemy list will be used to check the battle result. If one role's HP point equals to 0, the alive status will be changed. If the alive one of those lists is 0. then the result checker will return to a combat result.

```
m_allPlayerRoleList.Add(_role1);
m_allPlayerRoleList.Add(_role2);
m_allEnemyList.Add(_role3);
m_allEnemyList.Add(_role4);

m_allRoleList.Add(_role1);
m_allRoleList.Add(_role2);
m_allRoleList.Add(_role3);
m_allRoleList.Add(_role4);
```

Code Snippet 9. Set up the list for the player role and the enemies.

The Combat All Roles' list will sort after the end of turn, each time, it will check the status of all the list members' alive status and resort those members still alive into the combat all roles list(m_sortRoleList).

```
private void SortRole()
{
  m_sortRoleList.Clear();
  for (int i = 0; i < m_allRoleList.Count; i++)
{
          if (m_allRoleList[i].AliveStatus == false)
          {
             m_sortRoleList.Add(m_allRoleList[i]);
          }
      }
m_sortRoleList.Sort(delegate (RoleBattle a, RoleBattle b) {
 return b.RoleSpeed.CompareTo(a.RoleSpeed); });
    }
```

Code Snippet 10. Sort all roles and enemy turn order.

These two methods control the turn order, every time the turn order moves to the next one, it will find the roles or enemy should be active.

```
public RoleBattle GetBattleRole()
    {
        RoleBattle _role = GetTurnRole();
        if (_role == null)
        {
           ++m_turnCount;
           turncount = m_turnCount;
           turncounttext.text = turncount.ToString();
           SortRole();
           _role = GetTurnRole();
        }
CircleT.transform.position = GameObject.Find("Roles/"+_role.Name).transform.position;
        _anim = GameObject.Find("Roles/" + _role.Name + "/role").GetComponent<Animator>();
        _turnrole = _role;
        return _role;
    }
```

Code Snippet 11. Check the turn order target and activate it.

This method will confirm the game character for that turn and move the sign circle representing the game order to the corresponding position.

```
protected RoleBattle GetTurnRole() {

  if (m_sortRoleList.Count <= 0)
    return null;

  RoleBattle _turnRole = m_sortRoleList[0];

  if (_turnRole.AliveStatus == true)
  {
    AttackEnd();
    m_sortRoleList.Remove(_turnRole);
    return GetTurnRole();
  }
  else {
    if (_turnRole.RoleT == RoleType.Player) {
      SkillPanelFunction(true);
    }
    else
    {
      SkillPanelFunction(false);
    }
    AttackEnd();
    m_sortRoleList.Remove(_turnRole);

    return _turnRole;
  }
```

Code Snippet 12. Check the target on the turn order.

This method will confirm the game character for the turn and remove the character that has nearly used up all its action points from the action list for the turn, and if it is a character that the player can control, the corresponding user interface panel will appear.

5.3.7 Combat and Result Check

Depending on the outcome of the game, different levels of the play will determine different bonus points.

```
    public void AttackEnd()
    {
        if (BattleWIN() == true)
        {
            EndBattle(1000 * _battlelevel);
        }
        if (BattleLose() == true)
        {
            EndBattle(-1000 * _battlelevel);
        }
}
```

Code Snippet 13. Check the combat status and represent results and rewards.

If the game results in a win, then 1000 times the current difficulty factor will be awarded, and if the game results in a loss, then the same reward will be deducted.

```
private bool BattleWIN()
    {
        bool _finish = true;
        for (int i = 0; i < m_allEnemyList.Count; i++)
        {
            if (m_allEnemyList[i].AliveStatus == false)
            {
                _finish = false;
            }
        }
        ///Debug.Log(_finish);
        return _finish;
    }

    private bool BattleLose()
    {
        bool _finish = true;
        for (int i = 0; i < m_allPlayerRoleList.Count; i++)
        {
            if (m_allPlayerRoleList[i].AliveStatus == false)
            {
                _finish = false;
            }
        }
        ///Debug.Log(_finish);
        return _finish;
    }
```

Code Snippet 14. Combat Status Check

By checking the status of the objects in different lists, the game comes to a final game ending of victory or defeat.

```
    private bool MovePointCheck(int _n) {
        if (selected != null) {
        movepoint  -= _n;
        if (movepoint >= 0)
        {
            return true;
        }
        else
            return false;
        }
        else
        {
            SelectWarining();
                return false;
        }
    }
```

Code Snippet 15. Move point check

Returning to the corresponding Boolean value by checking whether the remaining number of points is greater than or equal to 0.

```
public void DoHeal()
    {
        RoleBattle _target = BattleManager.Instance.TargetedU(RoleT);

        if (_target.RoleT == RoleType.Player)
        {
            if (_target.AliveStatus == false)
            {
                if (_target.RoleHPNow >= _target.RoleTotalHP)
                    {
```

```
                Debug.Log(_target.RoleTotalHP);
                GameObject.Find("BattleUICanvas/InfoPanel/InfoPanelS/" + _target.Name +
"/HP").GetComponent<Text>().text = _target.RoleTotalHP.ToString();
                GameObject.Find("BattleUICanvas/InfoPanel/InfoPanelS/" + _target.Name +
"/Slider").GetComponent<Slider>().value = _target.RoleTotalHP;
                }
            else
                {
                _target.BeAttack(RoleAttack);
                if (_target.RoleHPNow > _target.RoleTotalHP) {
                    _target.SetHPFull();
                }
                GameObject.Find("BattleUICanvas/InfoPanel/InfoPanelS/" + _target.Name +
"/HP").GetComponent<Text>().text = _target.RoleHPNow.ToString();
                GameObject.Find("BattleUICanvas/InfoPanel/InfoPanelS/" + _target.Name +
"/Slider").GetComponent<Slider>().value = _target.RoleHPNow;
                }
        }
        else
        {
            Debug.Log("a dead target");
            Debug.Log(_target.RoleHPNow);
        }
    }
    else
    {
        Debug.Log("select a player role!");
    }}
```

Code Snippet 16. An example for skill method implementation

Different combat options are implemented in different methods. For example, the Healing option, which is implemented after confirming the selected enemy role and the player role’s status, calls the corresponding method, then changes the result values to the user interface.

5.3.8 Mouse Click to Select the GameObject

In Unity, identifying an item with a collision component is greatly simplified by a method of colliding with a ray from the camera in the direction indicated by the mouse.

```
void Update()
      {

         if (Input.GetMouseButtonDown(0))
         {
            if (UIselected != null)
            {
               UIselected.GetComponent<Image>().color = Color.white;
            }
            else
            {}

            RaycastHit hitInfo = new RaycastHit();
            bool hit = Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition),
out hitInfo);
            if (hit)
            {
               UIselected = hitInfo.transform.gameObject;
               UIselected.GetComponent<Image>().color = Color.red;
               //Debug.Log(UIselected.name);
               if (UIselected.name != "Save1" && UIselected.name != "Save2" &&
UIselected.name != "Save3")
               {
                  GameObject.Find("/Main
Camera/GameCanvas/LoadPanel/DataDescription").GetComponent<Text>().text =
UIselected.name;
                  _levelname = UIselected.name;
                  _SaveName = UIselected.name;
               }
               else {
               PlayerDatabase playerDatabase = new PlayerDatabase();
               string _des = playerDatabase.ReadPlayerName(UIselected.name);
               GameObject.Find("/Main
Camera/GameCanvas/LoadPanel/DataDescription").GetComponent<Text>().text= _des;
               _levelname = UIselected.name;
               _SaveName = UIselected.name;
               }
            }
            else
            {
```

```
            Debug.Log("No hit");
        }}}
```

Code Snippet 17. UI Object clicker script

Once the player presses the mouse, the program calls the method to fire a beam of light in the direction of the mouse through the location of the camera, and when it hits a game object with a collision volume, returns the object as the selected object, and then changes the corresponding selection name to the selected object name.

5.3.9 Player Data Load and Save

In Unity, Playerprefs provides a class for local persistent storage and reading of data. That means that data saved to Playerprefs can be transferred between different scenes, rather than using a do not destroy GameObject transfer data between different scenes. It can be used to save some simple localized data easily. The data is saved to Playerprefs as key-value pairs form and can be read by the key name, and when the value does not exist, the default value is returned.

JSON save files can be read and written directly form the right file path.

```
public void ReadData(string _savename)
    {
        playerdata = JsonMapper.ToObject(File.ReadAllText(Application.dataPath +
"/Data/Save/" + _savename + ".json"));
        PlayerPrefs.SetString("SaveName", _savename);
        PlayerPrefs.SetString("playerName", playerdata[0].ToString());
        PlayerPrefs.SetInt("money", int.Parse(playerdata[1].ToString()));
        PlayerPrefs.SetInt("Level", int.Parse(playerdata[2].ToString()));
        PlayerPrefs.SetString("Role1", playerdata[3][0][0].ToString());
        PlayerPrefs.SetInt("Level1", int.Parse(playerdata[3][0][1].ToString()));
        PlayerPrefs.SetString("Role2", playerdata[3][1][0].ToString());
        PlayerPrefs.SetInt("Level2", int.Parse(playerdata[3][1][1].ToString()));
        PlayerPrefs.SetInt("BattleLevel", 1);
        SceneManager.LoadScene(1);
    }
```

Code Snippet 18. Read the JSON file

The player can select the specific save files by name. Then, all the necessary Playerprefs data will save to that file. A save file or save path cannot be null, or the game should create the files in this folder.

```
      public void SaveData(string _savename) {
        //intialNewGame();
RoleData rd1 = new RoleData(PlayerPrefs.GetString("Role1"), PlayerPrefs.GetInt("Level1"));
RoleData rd2 = new RoleData(PlayerPrefs.GetString("Role2"), PlayerPrefs.GetInt("Level2"));
          roles.Add(rd1);
          roles.Add(rd2);

          inv = GameObject.Find("ItemDatabase").GetComponent<Inventory>();
          for (int i = 0; i < inv.items.Count;i++)
          {
             InventoryData IS = new InventoryData(i, inv.items[i].itemId);
             iSlots.Add(IS);
          }

PlayerData newplayer = new PlayerData(PlayerPrefs.GetString("playerName"),
PlayerPrefs.GetInt("money"), PlayerPrefs.GetInt("Level"),roles, iSlots);
          playerdata = JsonMapper.ToJson(newplayer);
          File.WriteAllText(Application.dataPath + "/Data/Save/" + _savename + ".json",
playerdata.ToString());
      }
   }
```

Code Snippet 19. Save to the JSON file

When the player chooses to start a new game, all the PlayerPrefs values should clear up, and default values should save to PlayerPrefs. The game will automatically save those data to a “Save1” JSON file. Then, the game will start and change to the Game Main scene.

```
public void intialNewGame(string _new)
{
          if (_new == null) {
             _new = "NewPlayer";
```

```
        }
        PlayerPrefs.DeleteAll();
        PlayerPrefs.SetString("SaveFile", "Save1");
        PlayerPrefs.SetString("playerName", _new);
        PlayerPrefs.SetInt("money", 1000000);
        PlayerPrefs.SetInt("Level", 1);
        PlayerPrefs.SetString("Role1", "New1");
        PlayerPrefs.SetInt("Level1", 1);
        PlayerPrefs.SetString("Role2", "New2");
        PlayerPrefs.SetInt("Level2", 1);
        PlayerPrefs.SetInt("BattleLevel", 1);
        ///SaveData("Save0");
        RoleData rd1 = new RoleData("New1", 1);
        RoleData rd2 = new RoleData("New2", 1);
        roles.Add(rd1);
        roles.Add(rd2);
    }
```

Code Snippet 20. Initial the Game.

After the player chooses to start a new game, the program automatically writes the preset initial values to the game.

6 TESTING

The testing process is always essential in the development work and can effectively help developers to find problems that have not been encountered during the development process. At the same time, developers experience the project from the view of users and receive more comprehensive early feedback to improve the project. This test of the project was done on a desktop computer with the Windows 10 system.

Because the settings of Unity development and Unity games are different, for this project, a Data folder with save files insider is needed.

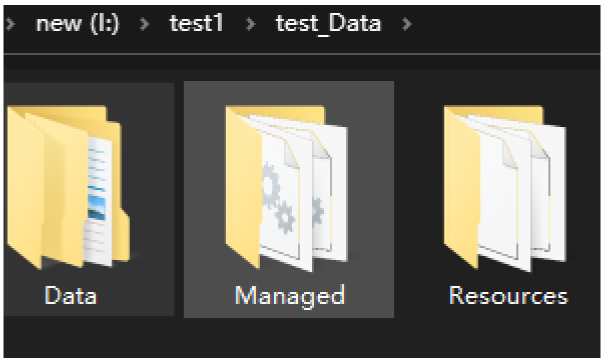


Figure 24. The Data folder with save files inside

6.1 Start Game Scene

All the functions were working well, but when calling a load game function for the first time, or starting a new game function, the game stuttered for a few seconds. It seems that performance optimization is needed.

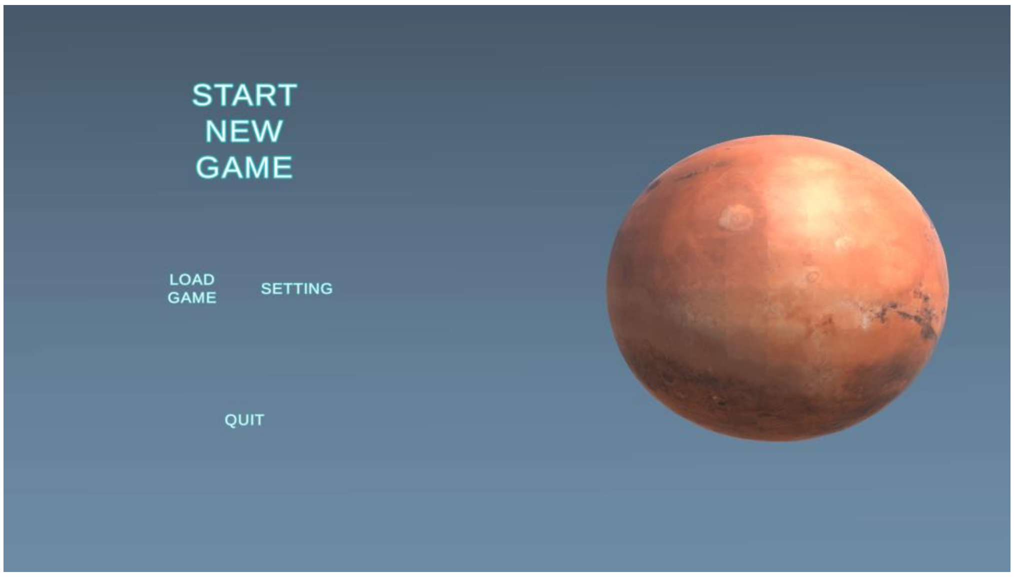


Figure 25. Start Game Scene test

6.2 Game main scene

Figure 26 is an example of game main scene interface.

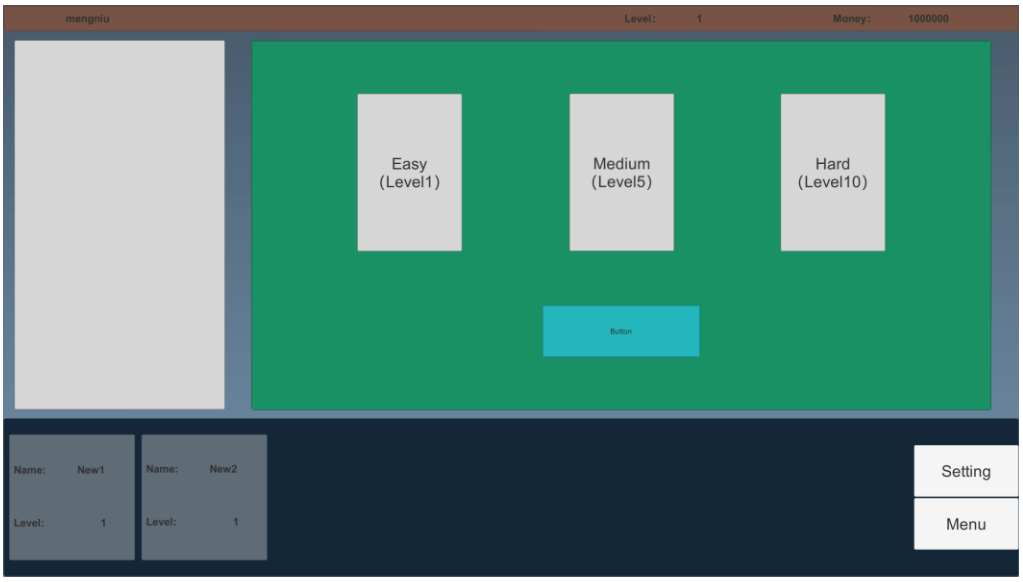


Figure 26. Start Game Scene test

When starting a new game with a player name “Mengniu”, everything worked well. A cutscene when the player changing scene is needed, switching directly to the next scene may be too crude. All the default data was correct and no exceptions.

6.3 Save and Load Scene

All the save and load functions worked well, the game can read and load the selected files. All the data can be saved to those files correctly.

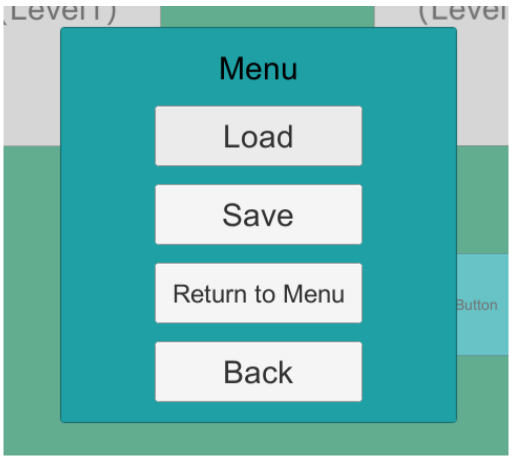


Figure 27. Menu panel view

6.4 Game Combat Scene

The functions of the game combat view worked well, when selecting different challenge levels, the enemy received different strengths or weaken. The rewards could be received correctly. The data can be saved to the saved files.

However, the design checks the results after the end of the turn, which means if there is a target still in the Sort List, the result view will active after the turn is finished. That should be optimized and improved.

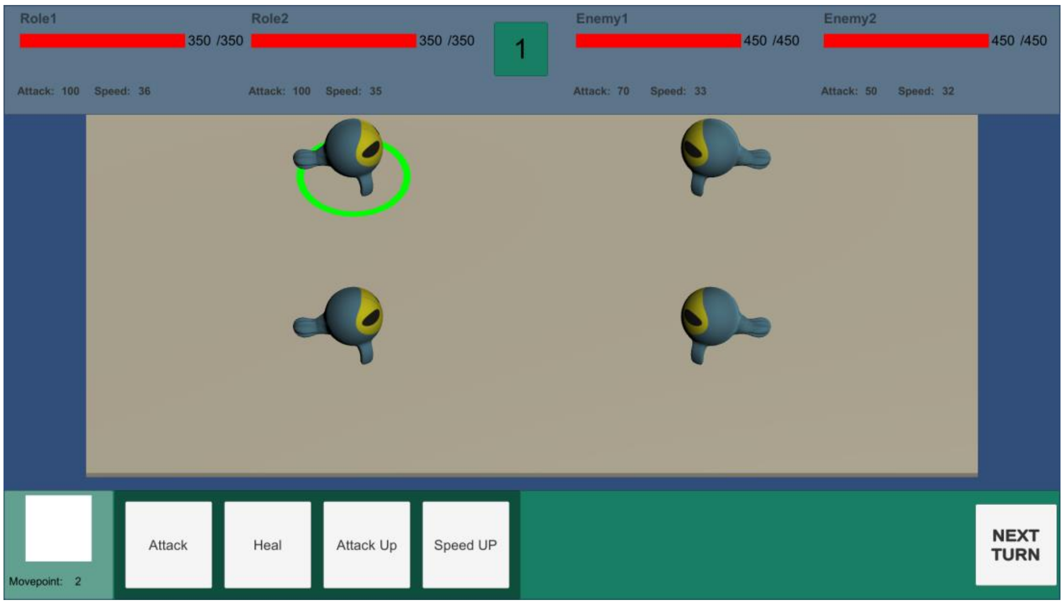


Figure 28. Combat scene view

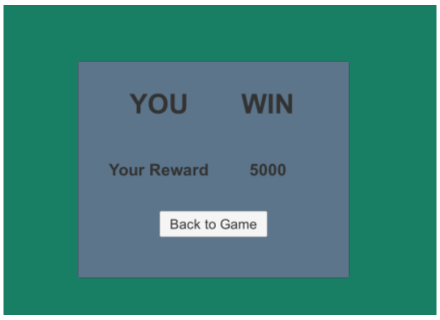


Figure 29. Combat result View

After confirming the game result and returning to the game main scene, the player will find that the acquired resources have been updated.

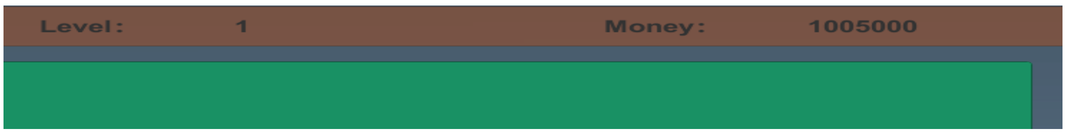


Figure 30. Rewards Panel View

7 CONCLUSIONS

The game project completes the plan of the paper, while this paper provides a detailed description and explanation of how to develop a Unity game and the technologies required. The three scenes were completed to make the game experience complete. Because of the easy-to-read JSON file design, gamers can make changes to the game files directly. Players can simply experience a turn-based strategy game, then save and load their game files.

As one of the most popular game development engines, Unity is open to those people who cannot afford an expensive professional game engine and very friendly help to various developers. Even though Unity is more suitable for lightweight, multi-platform game development. Because many functions are not provided directly, this causes a lot of difficulties in learning and development.

The Unity3D engine updates very soon, quick updates include bug fixing and new features bring great convenience for developers. However, all the game engine files need to be downloaded again instead of through incremental updates. But the project files moving between different engine versions caused much trouble such as compile errors.

Although the project accomplished its design objectives, it was not good enough as a professional game. The content of the game could include more different enemy types and game maps could be added, more mission types and challengeable objectives could be designed, more equipment types and more equipment effects should be provided by the equipment system, and at the same time, the base construction content could be further developed. Finally, the art design style of the game needs to be unified, and a more beautiful user interface and scene design will be completed in future development.

REFERENCES

/1/ Williams, Andrew. 2017. History of Digital Games: Developments in Art, Design and Interaction. 10.1201/9781315715377.

/2/ China Game Industry Report 2018. Accessed 29.04.2021.

http://www.cgigc.com.cn/gamedata/20752.html

/3/ Strategy video game, Wikipedia. Accessed 29.04.2021.

https://en.wikipedia.org/wiki/Strategy_video_game

/4/ Game design document, Wikipedia. Accessed 29.04.2021.

https://en.wikipedia.org/wiki/Game_design_document

/5/ Game engine, Wikipedia. Accessed 29.04.2021.

https://en.wikipedia.org/wiki/Game_design_document

/6/ Unity, Wikipedia. Accessed 29.04.2021.

https://en.wikipedia.org/wiki/Unity_(game_engine)

/7/ Blender. Accessed 29.04.2021.

https://www.blender.org/

/8/ The JavaScript Object Notation (JSON). Accessed 29.04.2021.

http://json.org/

/9/ LitJSON Accessed 29.04.2021 https://litjson.net/

/10/ Unity User Manual. Accessed 29.04.2021.

https://docs.unity3d.com/Manual/index.html