
The 3D Role-Playing Game

Qi Fu

B.Sc.(Hons) in Software Development

APRIL 29, 2018

Final Year Project

Supervisor : Gerard Harrison

Department of Computer Science and Software Development

Galway-Mayo Institute of Technology (GMIT)



Contents

1	Introduction	4
2	Context	6
2.1	Application Information	6
2.1.1	Expected Objectives	7
3	Methodology	8
4	Technology Review	9
4.1	Scripts of Game Objects	9
4.2	Monsters AI	15
4.3	Inventory System	18
4.4	Data	20
5	System Design	22
6	System Evaluation	26
7	Conclusion	28

About this project

Abstract This is a basic Action Role Playing Game that allows users to manipulate characters and complete a set of actions including moves and attacks. It has life value calculation system to reflect real-time life value and also allows the character to interact with game objects.

This application is designed by Unity 3D, the main program languages are C# and Java. It allows player manipulates character with the keyboard to control its movement and achieve a set of actions. The Character can be moved by using keyboard "w-a-s-d" to control direction and arrow key "up-down-left-right" can adjust the direction of view. The key "J" is Attack order and key "F" can interact with game objects like communication.

This game not only has the player character, it also has monster enemy characters. Monster characters usually patrol on the specific route, once monsters see there is a player character in their detect range, they will catch the player and if the player out of their range, monsters will back to patrol model. This function is an advanced feature, it achieves basic monsters' AI controller.

The player character has UI panel to display the information of the character and current situation especially the rest of life value, name, and level of character.

Authors Qi Fu

Chapter 1

Introduction

PC games, also known as computer games or personal computer games, are video games played on a personal computer rather than a dedicated video game console or arcade machine. Their defining characteristics include a more diverse and user determined gaming hardware and software, and a generally greater capacity input, processing, and video output.

Home computer games became popular following the video game crash of 1983 leading to the era of the "bedroom coder". In the 1990s, PC games lost mass-market traction to console games before enjoying a resurgence in the mid-2000s through digital distribution.

Newzoo, reports that the PC gaming sector is the third largest (and estimated in decline), with the consoles second largest, and across all platforms as of 2016, 2.2 billion gamers generate US\$101.1 billion in revenue (i.e. all numbers exclude hardware costs), and "Digital game revenues will account for \$94.4 billion or 87% of the global market. Mobile is the most lucrative segment, with smartphone and tablet gaming growing 19% year on year to \$46.1 billion, claiming 42% of the market. In 2020, mobile gaming will represent just more than half of the total games market. China expected to generate \$27.5 billion, or one-quarter of all revenues in 2017." PC is considered synonymous (by them and others) with IBM PC compatible systems; while mobile computers – smartphones and tablets, such as those running Android or iOS – are also personal computers in the general sense. The "APAC" region is estimated to generate \$46.6 billion in 2016 or 47% of total global game revenues (note, not only "PC" games). China alone accounts for half of APAC's revenues, reaching \$24.4 billion, cementing its place as the largest games market in the world, ahead of the US's anticipated market size of \$23.5 billion. China is expected to have 53% of revenues from mobile in 2017 (46% in 2016). The uncoordinated nature of the PC game market and its lack of physical media make precisely assessing its size difficult.[?]

Types of Game

RPG: Role-playing Game is a game in which players assume the roles of characters in a fictional setting. Players take responsibility for acting out these roles within a narrative, either through literal acting or through a process of structured decision-making or character development. Actions taken within many games succeed or fail according to a formal system of rules and guidelines.

WEG: Web Game(Browser Game) is a computer game that is played over the Internet using a web browser. Browser games can be run using standard web technologies or browser plug-ins. The creation of such games usually involves the use of standard web technologies as a frontend and other technologies to provide a backend. Browser games include all video game genres and can be single-player or multiple-player. Browser games are also portable and can be played on multiple different devices, web browsers, and operating systems.

ACT: Action Game is a video game genre that emphasizes physical challenges, including hand-eye coordination and reaction time. The genre includes diverse subgenres such as fighting games, shooter games and platform games which are widely considered the most important action games, though multiple-player online battle arena and some real-time strategy games are also considered to be action games. In an action game, the player typically controls the protagonist or avatar. The avatar must navigate a level, collecting objects, avoiding obstacles, and battling enemies with various attacks. At the end of a level or group of levels, the player must often defeat a boss enemy that is more challenging and often larger than other enemies. Enemy attacks and obstacles deplete the avatar's health and lives, and the player receives a Game over when they run out of lives. Alternatively, the player wins the game by finishing a sequence of levels. But some action games, usually arcade games, are unbeatable and have an indefinite number of levels; and the player's only goal is to maximize their score by collecting objects and defeating enemies.

Chapter 2

Context

GitHub Link: github.com/QiFuChina/RPG. It contains:

- Application package.
- Documents of application.
- Assets Package.
- Source code of application.

Video Line: [here](#)

2.1 Application Information

- Design Platform : Windows Computer
- Design Technology : Unity 3D
- Design Model : Single client service
- Game Type : Role-Playing Game
- Design Inspiration : It from a Chinese Wuxia online game.

- Manipulate : The Character can be moved by using keyboard "w-a-s-d" to control direction, "up-down-left-right" can adjust the direction of view. The key "J" is Attack order and key "F" can interact with game objects like communication.
- Model : Character model relative with animation, player press keyboard then the model will reflect a set of animation as feedback
- Objects : Enemy: The enemy has "enemy" target that different with the character when closed to the character it will attack Trap: These traps have "enemy" target and cause side-effect to the character when trigger them Scene: To allow scenes switch when the character dies.

2.1.1 Expected Objectives

Basic Objectives:

- Completely moving functions and player character status display scripts.
- Attack and hurt function.
- Monster enemies controller.

Advanced Objectives:

- Game data save and load.
- Monsters AI tree.
- Animation controller.
- Player inventory system.

Chapter 3

Methodology

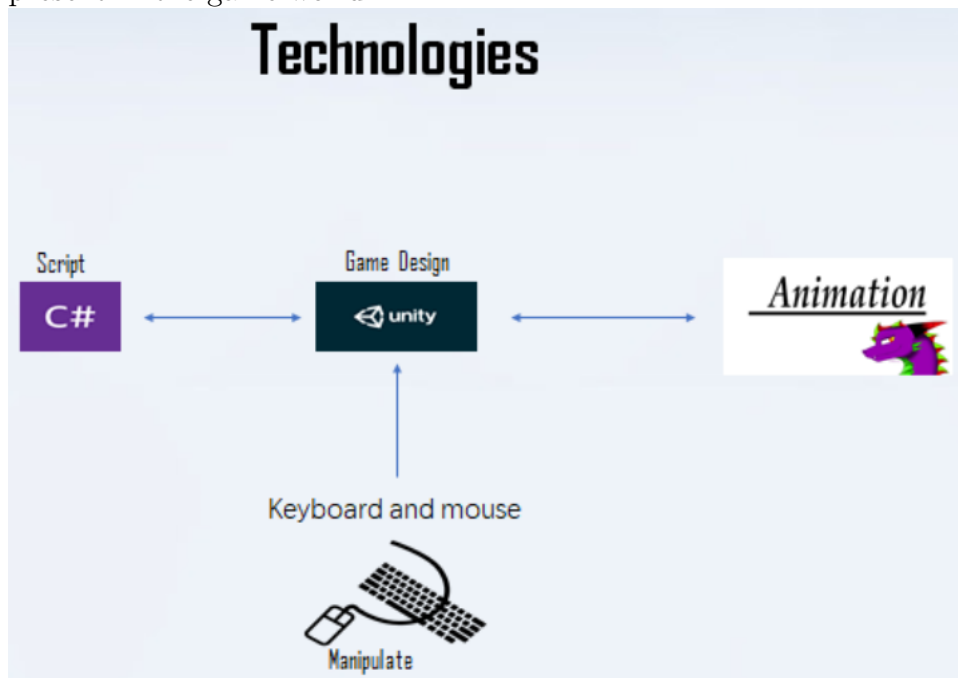
- Agile / incremental and iterative approach to development.
This project aimed at basic functions of the regular role-playing game at first, and then develop the advanced functions after basic functions finished.
- What about validation and testing?
It can be worked successfully in the windows 10 64bit system.
- If team based, did you use GitHub during the development process.
No, this is an individual project, but I am looking forward using GitHub during team development process.
- Selection criteria for algorithms, languages, platforms, and technologies.
C# is the wide language that is used in Unity 3D game development industry so that is the reason why I used C# and Unity 3D to develop this project.

Chapter 4

Technology Review

4.1 Scripts of Game Objects

This graph is the structure of the application, Unity is the design platform and it contains C# scripts and animation controller. The command will be inputted from keyboard and application process it to present in the game world.



Description :

Unity is a cross-platform game engine developed by Unity Technologies[2], which is primarily used to develop both three-dimensional and

two-dimensional video games and simulations for computers, consoles, and mobile devices. First announced only for OS X at Apple's Worldwide Developers Conference in 2005, it has since been extended to target 27 platforms[3][4]. Six major versions of Unity have been released. For a list of games made with Unity, visit [List of Unity games](#).

Unity is a multipurpose game engine that supports 2D and 3D graphics, drag-and-drop functionality and scripting using C#. Two other programming languages were supported: Boo, which was deprecated with the release of Unity 5[5] and JavaScript which started its deprecation process in August 2017 after the release of Unity 2017.1[6]. The engine targets the following graphics APIs: Direct3D on Windows and Xbox One; OpenGL on Linux, macOS, and Windows; OpenGL ES on Android and iOS; WebGL on the web; and proprietary APIs on the video game consoles. Additionally, Unity supports the low-level APIs Metal on iOS and macOS and Vulkan on Android, Linux, and Windows, as well as Direct3D 12 on Windows and Xbox One. Within 2D games, Unity allows importation of sprites and an advanced 2D world renderer. For 3D games, Unity allows specification of texture compression, mipmaps, and resolution settings for each platform that the game engine supports,[3] and provides support for bump mapping, reflection mapping, parallax mapping, screen space ambient occlusion (SSAO), dynamic shadows using shadow maps, render-to-texture and full-screen post-processing effects⁷. Unity also offers services to developers, these are: Unity Ads, Unity Analytics, Unity Certification, Unity Cloud Build, Unity Everyplay, Unity IAP, Unity Multiplayer, Unity Performance Reporting and Unity Collaborate. Completely moving functions and player character status display scripts.

Here is the technology of character moving functions:

```

3 references
public class Player : MonoBehaviour {
    [Header("Player Attributes")]
    8 references
    public int hp;
    [Range(0,100)]
    5 references
    public int hpMax=100;

    [Header("Player UI")]
    1 reference
    public Text hpText;
    3 references
    public Image hpBar;
    1 reference
    public Text Level;
    0 references
    public int i=0,l=1;

    5 references
    public Animator anim;
    2 references
    public Collider AtkCollider;

    5 references
    private Player m_Player;
    2 references
    public Vector3 offset;
    2 references
    private float _pointY;
    0 references
    public float Speed = 0.3f;

```

These variables are player character attributes

```

void Start () {
    m_Player = GameObject.Find("Player").GetComponent<Player>();
    offset = transform.position- m_Player.transform.position;
    anim=GetComponent<Animator>();
    hpText.text="I'm Player";
    hp =hpMax;
    hpBar.fillAmount=(float)hp/(float)hpMax;
    Level.text="1";
}

```

Figure 1

After game sense be loaded, the start function will be executed first to find the game object that name is "Player" and access its position, then set the default name, healthy value, and level to the character.

```

void Update () {
    _pointY = m_Player.transform.eulerAngles.y;
    //Debug.Log(_pointY);//0~360
    Quaternion rotation = Quaternion.Euler(0, _pointY, 0);
    transform.position = Vector3.Lerp(transform.position, m_Player.transform.position +
    [rotation * offset],Time.deltaTime*damping);
}

```

Figure 2

The update function will be updated every frame to detect the play character status and the first thing is making the camera look at the character in the fixed degree.

```

transform.LookAt(m_Player.transform.position);

    if (Input.GetKey (KeyCode.W)) {
        gameObject.GetComponent<Transform> ().Translate
        (Vector3.forward * 0.05f, Space.Self);
    }
    if (Input.GetKey (KeyCode.S)) {
        gameObject.GetComponent<Transform> ().Translate
        (Vector3.back * 0.05f, Space.Self);
    }
    if (Input.GetKey (KeyCode.A)) {
        gameObject.GetComponent<Transform> ().Translate
        (Vector3.left * 0.05f, Space.Self);
    }
    if (Input.GetKey (KeyCode.D)) {
        gameObject.GetComponent<Transform> ().Translate
        (Vector3.right * 0.05f, Space.Self);
    }
}

//turn
if (Input.GetKey (KeyCode.LeftArrow)) {
    gameObject.GetComponent<Transform> ().Rotate (0f, -2f, 0f);
}
if (Input.GetKey (KeyCode.RightArrow)) {
    gameObject.GetComponent<Transform> ().Rotate (0f, 2f, 0f);
}

```

Figure 3

After camera look at the player, the update function will be called each frame, once player input commands through the keyboard, the relative function will be executed.

From this picture, we know "w-s-a-d" and "←--→" keys allow the character to move.

Attack and hurt function.

```

if (Input.GetKey (KeyCode.J)) {
    anim.SetBool ("Attack", true);
    AtkCollider.GetComponent<SphereCollider>().enabled = true;
} else {
    anim.SetBool ("Attack", false);
    AtkCollider.GetComponent<SphereCollider>().enabled = false;
}
hpBar.fillAmount=(float)hp/(float)hpMax;

```

Figure 4

The key "J" is the attack command that links the animator controller and object collider which connect with the player character. If "J" be triggered, the animation of "Attack" will play and the atksphere collider will enable to be detected.

The hp expression make sure the healthy value keeps updating all the time.

```

0 references
void OnTriggerEnter(Collider col){
    if(col.tag=="AtkSphereEnemy"){
        if(hp>0){
            anim.SetTrigger("hit");
            hp=Mathf.Clamp(hp-5,0,hpMax);
            print("player being hit");
        }
        if(hp<=0){
            anim.SetBool("die",true);
            print("player die");
            exitButton.SetActive(true);
            this.enabled=false;
        }
    }
}
hpBar.fillAmount=(float)hp/(float)hpMax;

```

Figure 5

This function relative to object collider, when player attack collider collides with the specific collider, the result will trigger animator controller and update the healthy value. Monster enemies controller.

<pre> 0 references public class MonsterAI : MonoBehaviour { [Header("Attributes")] 7 references public int hp; [Range(0,10)] 3 references public int hpMax=10; [Header("UI")] 1 reference public Text hpText; 1 reference public Slider hpSlider; </pre>	<pre> [Header("Component")] 2 references private Animator anim; 0 references private Collider col; 3 references private Collider atkSphereEnemy; 2 references private GameObject AI; </pre>
--	---

These variables are monsters attributes.

```

private void Awake () {
    hp=hpMax;
    anim=GetComponent<Animator>();
    // animation = gameObject.GetComponent<Animation>();
    // animation.clip = idleClip;
    // col=GetComponent<CapsuleCollider>();
    atkSphereEnemy=GetComponentInChildren<SphereCollider>();
    AI=transform.Find("AI").gameObject;
}

```

Figure 6

The awake function will execute when program initialized. It will awake the game component "AI" that binding with monster objects.

```
0 references
void Update () {
    hp=Mathf.Clamp(hp,0,10);
    hpText.text=hp+"/"+hpMax;
    hpSlider.value=(float)hp/(float)hpMax;
}
```

Figure 7

Initialized monsters healthy value every frame and response to monsters canvas.

```
0 references
void OnTriggerEnter(Collider col){
    if(col.tag=="AtkSphere"){
        hp-=2;
        // anim.SetBool ("atk", true);
        print("player atk");
        // anim.SetTrigger("hit");
        // animation.clip=atkClip;
        // animation.Play();
        // animation.clip=idleClip;
        // animation.Play();
        if(hp>0){

        }
        else{
            Die();
            AI.SetActive(false);
            // anim.SetBool ("atk", false);
        }
    }
}
```

```
void Die(){
    anim.SetTrigger("die");
    //col.enabled=false;
    Destroy(gameObject,1.0f);
    //i=i+5;
}
```

Once the trigger function detects monsters collide with other objects which target is "AtkSphere", monsters healthy will be decreased. When the healthy value under 0, the object will be destroyed.

```
0 references
public void AtkStartEnemy(){
    //animation.clip=atkClip;
    //animation.Play();
    print("atk start");
    atkSphereEnemy.enabled=true;
}
0 references
public void AtkStopEnemy(){
    print("atk stop");
    atkSphereEnemy.enabled=false;
}
#endregion
```

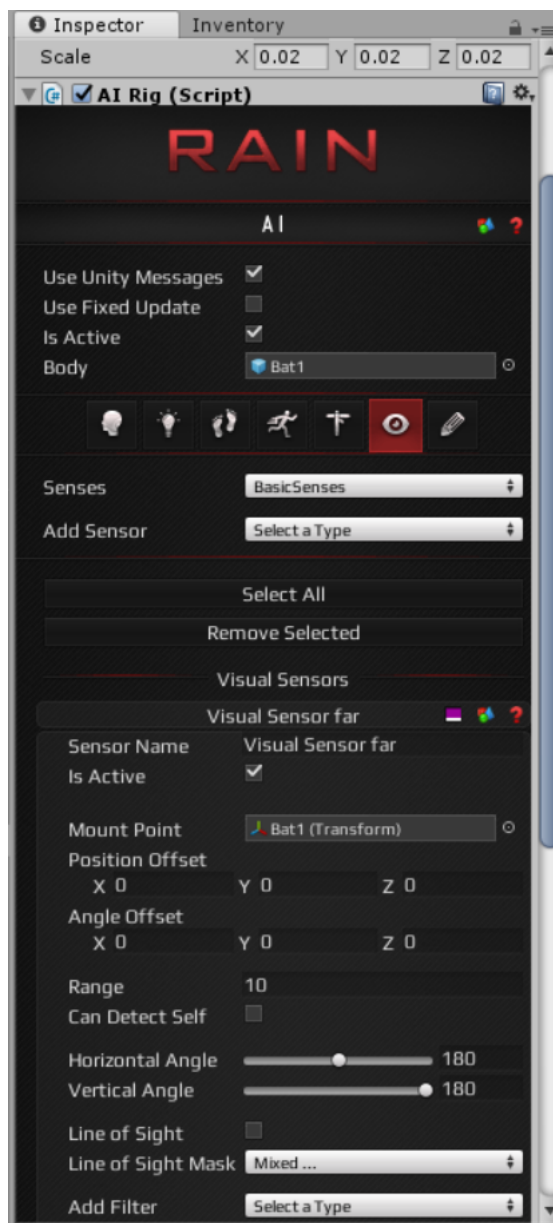
Figure 8

This function is attack function, when monsters collide with character then attack function will call attack animator and enable the sphere collider to make game update monsters and player healthy value.

4.2 Monsters AI

This is a tool of monsters AI controller.

- After downloading the RAIN package, import it and open it on the Unity edit row, selecting one object and click RAIN→Create New→AI to add the AI component to the target object.

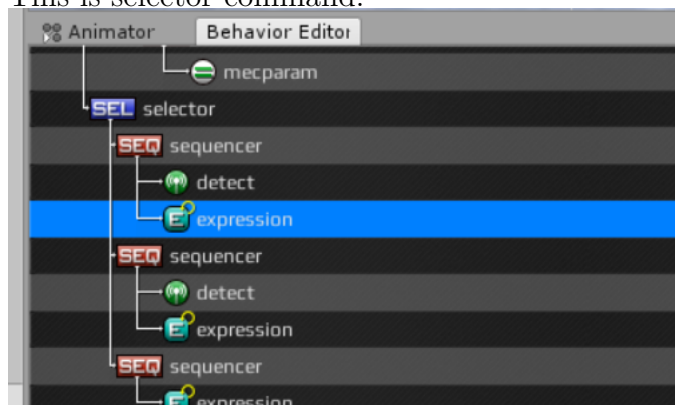


- This component has two visual sensors: far and near. These sensors make monsters can "see" the player character if they are close.
- The AI tree sets a list of condition to make monsters can execute correct performance under different situations.

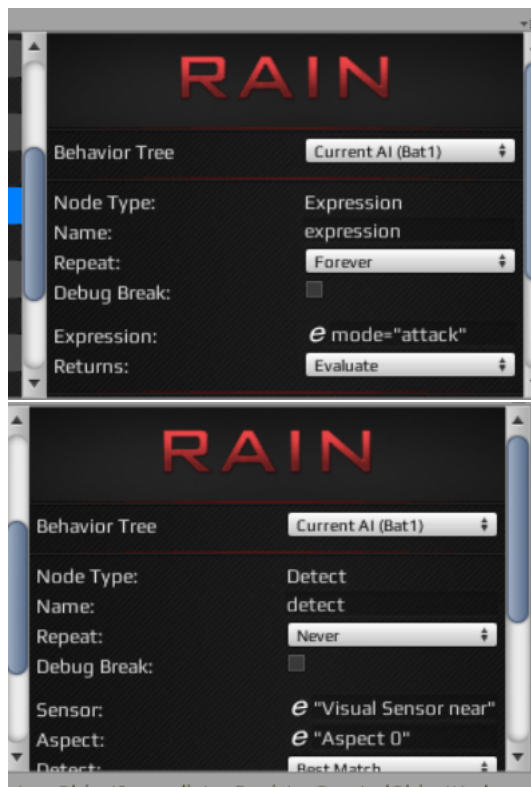


The constraint command is conditional judgment statement. To set a key value of the condition previously and create "waypointpatrol", once condition was triggered then relative constraint commands will be executed. And the same level of constraints are parallel relationship it means one constraint can break another constraint.

- This is selector command.



The selector will select one sub sequencer when the triggered condition be executed.

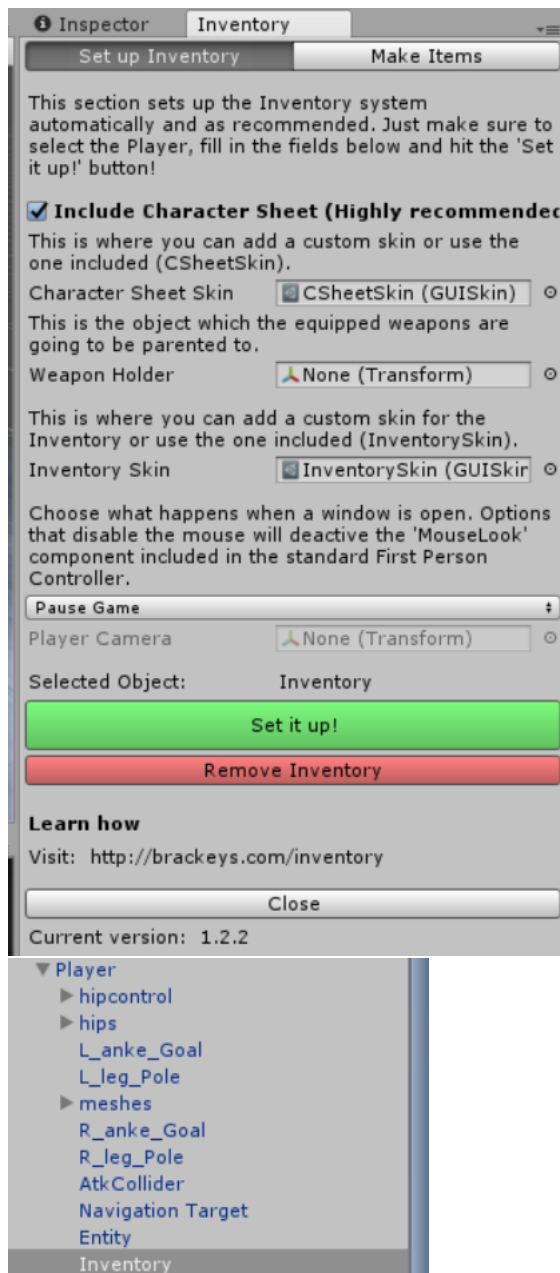


It is an example of attack command, we can see when the near of visual sensor detect the player character, the expression value will become attack that means the animator will be called, then it will call animation to play attack function and atksphere will enable to calculate the damage.

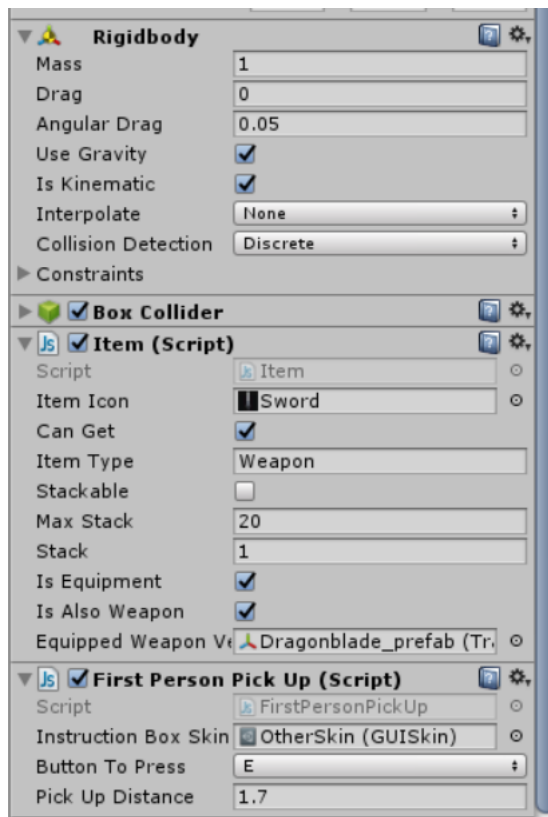
4.3 Inventory System

This is a tool of character inventory system.

- After download the Inventory System package, import it and open it on the Unity edit row, selecting one object and click Windows→Inventory to add the Inventory System component with the target object and just click suit it up.



After that, select a weapon object and add components "Rigidbody", "BoxCollider", "Item" and "First Person Pick Up" scripts. In the weapon object, changing these attributes to make weapon can be picked.



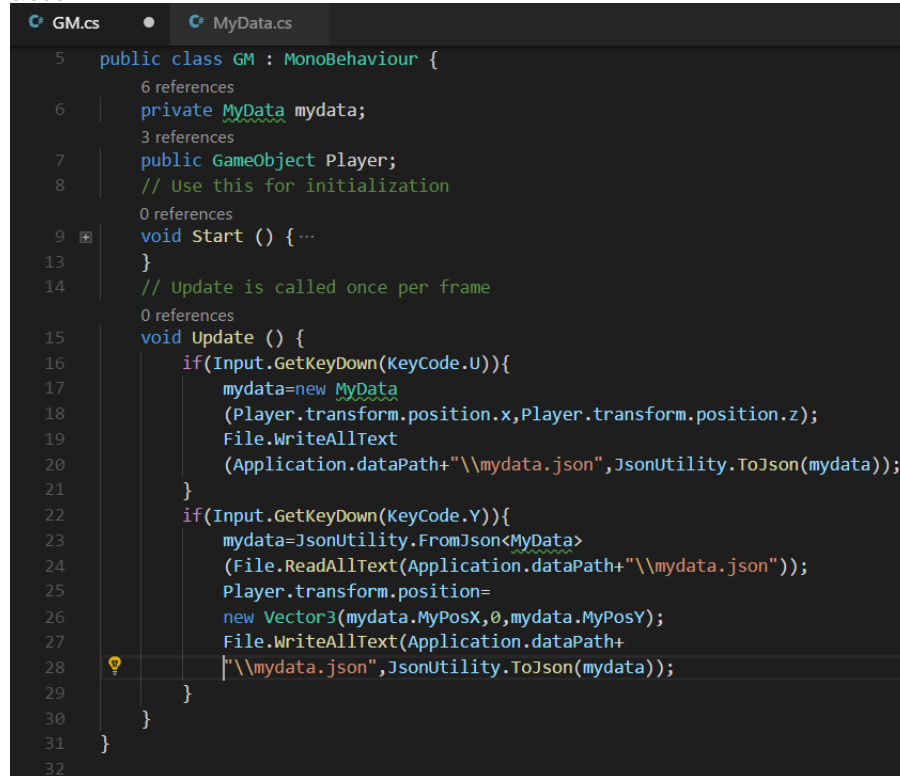
4.4 Data

These scripts manage the game data function.

```

MyData.cs • GM.cs
1
2  [System.Serializable]
   3 references
3  public class MyData {
   2 references
4      public float MyPosX;
   2 references
5      public float MyPosY;
   1 reference
6  public MyData( float myPosX, float myPosY) {
7      this.MyPosX=myPosX;
8      this.MyPosY=myPosY;
9  }
10 }
```

The Data.cs file created a class to store the position of the player character.



```

5 public class GM : MonoBehaviour {
6     private MyData mydata;
7     public GameObject Player;
8     // Use this for initialization
9     void Start () { ...
13 }
14 // Update is called once per frame
15 void Update () {
16     if(Input.GetKeyDown(KeyCode.U)){
17         mydata=new MyData
18         (Player.transform.position.x,Player.transform.position.z);
19         File.WriteAllText
20         (Application.dataPath+"\\mydata.json",JsonUtility.ToJson(mydata));
21     }
22     if(Input.GetKeyDown(KeyCode.Y)){
23         mydata=JsonUtility.FromJson<MyData>
24         (File.ReadAllText(Application.dataPath+"\\mydata.json"));
25         Player.transform.position=
26         new Vector3(mydata.MyPosX,0,mydata.MyPosY);
27         File.WriteAllText(Application.dataPath+
28         "\\mydata.json",JsonUtility.ToJson(mydata));
29     }
30 }
31 }
32

```

This GM.cs file used the function that in the Data.cs and defined a variable to get the player character.

Functions which inside in update are saved and loaded commands, users press "U" to get current position and "Y" to rewrite the player position.

The expression

"mydata=new MyData(Player.transform.position.x,Player.transform.position.z);"

will get position and save them to the variable mydata as Jason format.

The expression

"File.WriteAllText(Application.dataPath+"//mydata.json",JsonUtility.ToJson(mydata));"

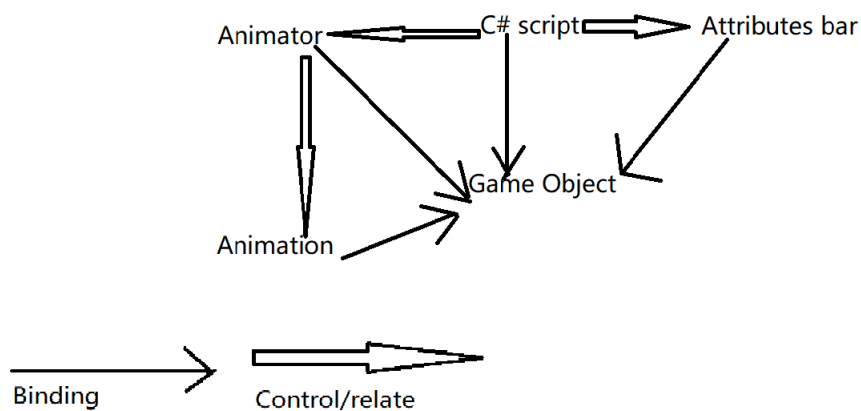
will acquire the player position when press "Y", and change data format from Jason to transform position then save it in the local directory, then apply them to the game object.

Chapter 5

System Design

The architecture of player character

The architecture of
Player Character

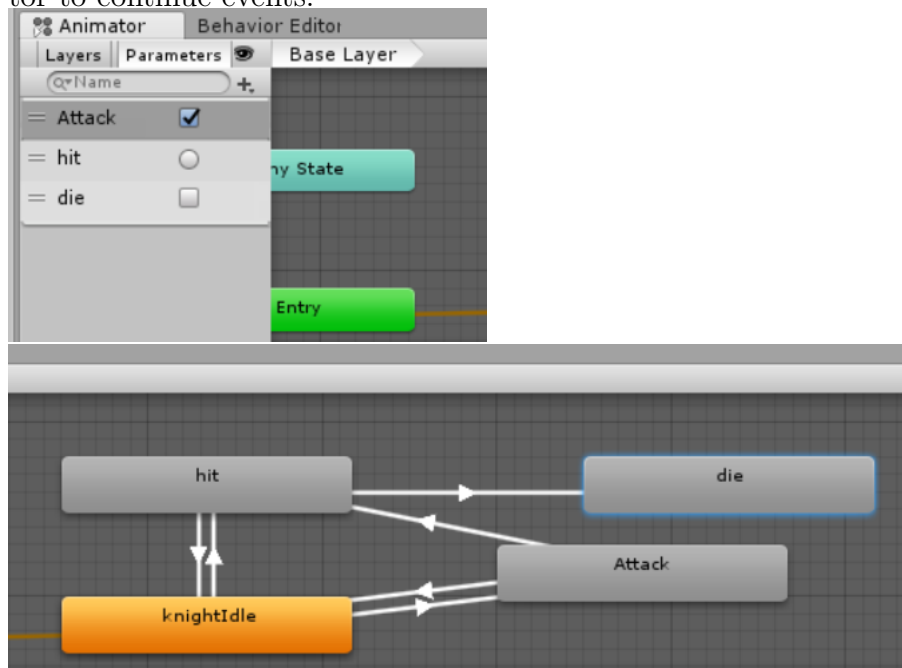


From the above picture, we know the player game object bindings with the rest of components and components also have relationships with themselves, results that will be generated from components will transfer to game object finally.

- C# scripts, animator, and animation have the control relationship. When users manipulate the character through the keyboard, C# scripts will send commands to animator, after the animator receives commands it will call the animation component, then the animation component

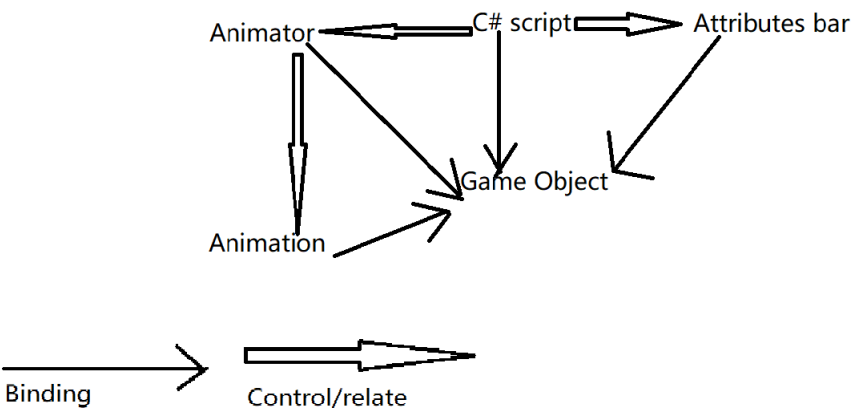
will call relative actions and transfer them to a game object.

- There is another condition that is the character be triggered by other game objects in the game, then the C# script also will call the animator to continue events.

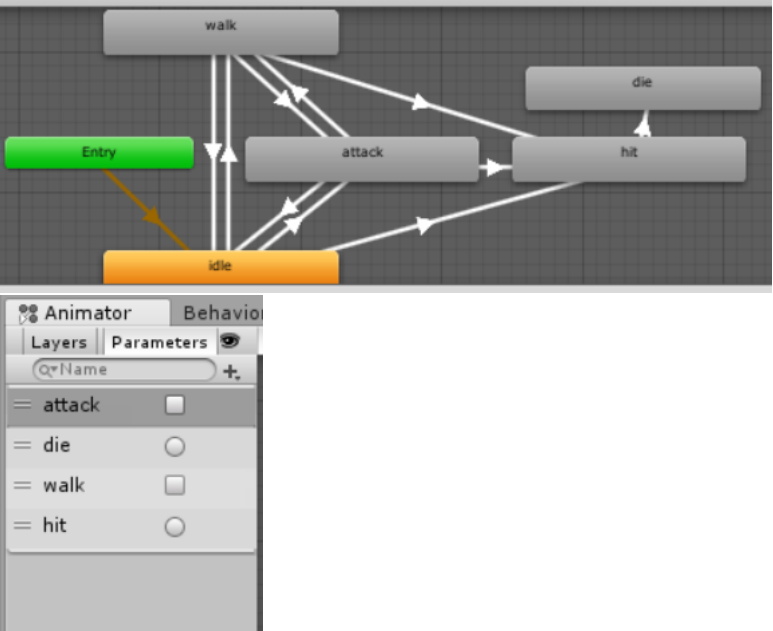


The architecture of monsters

Monsters architecture



The monster relationships are similarly with the player character. The animator controller in the below:



From these above pictures, we know each animation status enable to trigger another and break current status. The white arrow is a transition to change

their status with the artificial operation.

These animators can be awakened by C# scripts to achieve animation status changing.

Chapter 6

System Evaluation

Size of application

- This game total size is 167M with 64-bit and 162M with 32-bit. The evaluation device information in the below.

The 64-bit application occupies about 28% CPU, 95MB of memory, and about 60% GPU when it runs. The 32-bit application has the similar consumption.

- From the above result, we can see the performance benchmarks of application is not good enough because it consumes excessive resource as a basic role-playing game.

The outcomes of application

- From the introduction, there are 7 objectives to work and now, the basic objectives almost finished except some functions can be built more perfect. The advanced objectives of Monster AI tree and animation controller all almost finished. The inventory system is successful to acquire weapons but cannot make the player get extra enhance effects. The game save and load function can read and save player position as Jason file then load it but not able to record all information about the player.

limitations

- The first limitation of the application is users cannot rotate the character view direction by moving the mouse. It is obvious that moving the mouse to change view better than press " $\leftarrow\rightarrow$ " keys.

- Another limitation is the inventory system, this is an asset package that was designed by another developer. I only worked the small part of the entire system that I understood.

Chapter 7

Conclusion

Overall

- To review the whole context, the aim of the application is to build a role-playing game that has complete basic functions to play and has some advanced functions to improve users experience. And basic functions are finished mostly, advanced functions still have some part need to complete.

The outcomes of the project

- The Keyboard can manipulate the player character.
- The character can play exist animation when relative commands are triggered. For example, the character can play attack animation when pressing "J".
- Monsters are able to play animation when they attacked.
- The value of healthy will update automatically.
- The character and monsters can attack each other when life value becomes 0, it will die and the game object will be destroyed from the screen.
- Monsters will obey the AI tree system to perform.
- The player character can pick a weapon from the ground and open bags to catch it.
- The save function can read the position and load it.

summary

This game just meets regular functions, some advanced functions like equipment effect, high intelligent enemies and more functions that exist in publish role-playing game are not created. A single play game should have full save and load functions, it can be saved in the local directory or cloud service either. Obviously, this application can become a game that has various functions and I am enjoying the experience of working this game.