Research on Key Technologies Base Unity3D Game Engine

Jingming XIE
Information Engineering Institute
Guangzhou Panyu Polytechnic College
Guangzhou, China
xjmadam@hotmail.com

***Abstract*—Game engine is the core of game development. Unity3D is a game engine that supports the development on multiple platforms including web, mobiles, etc. The main technology characters of Unity3D are introduced firstly. The component model, event-driven model and class relationships in Unity3D are analyzed. Finally, a generating NPCs algorithm and a shooting algorithm are respectively presented to show common key technologies in Unity3D.**

***Index Terms*—game engine, Unity3D, NPC algorithm, shooting algorithm**

I. UNITY3D GAME ENGINE

A game engine provides a main framework and common functions for developing games, which is the core of controlling games. Since the first advent of the Doom game engine in 1993, game engine technology has experienced nearly 20 years of evolution. Game engine initially only supported 2D, now fully supports 3D, lifelike images, massively multiplayer online game, artificial intelligence and mobile platforms. Some representative game engines are Quake [1], Unreal Tournament [2], Source [3], BigWorld [4] and CryENGINE [5], etc. The internal implementation techniques in Game engines have some differences, but their ultimate goal is the same to improve the efficiency of game development. To comprehensively grasp the general development ideas of a game, it is necessary for choosing a typical game engine to study deep.

Unity3D is a popular 3D game engine in recent years, which is particularly suitable for independent game developers and small teams. It mainly comprises eight products such as the Unity, Unity Pro, Asset Server, iOS, iOS Pro, Android and Android Pro [6]. Without writing complex codes, programmers can quickly develop a scene by using the visual integrated development environment of Unity3D. In the popular iPhone game list, the games developed by Unity3D take a large proportion, such as Plants vs. Zombies, Ravensword:The Fallen King. In particular, Unity3D also provides the Union and Asset Store selling platforms for game developers.

Unity3D has special advantages in easily programming a game. For example, platform-related operations are encapsulated in its internal, the complex game object-relations are managed by different visual views, and JavaScript, C # or Boo scripting languages are applied to program a game. A script program will be automatically compiled into a .NET DLL file, so the three scripting languages, in essence, have the same performance, their execution speed is 20 times faster than traditional JavaScript. These script languages have good cross-platform ability as well. That means developers can deploy games on different platforms such as Windows, Mac, Xbox 360, PlayStation 3, Wii, iPad, iPhone and Android. In addition, games can run on the Web by installing a plug-in.

Another feature of the Unity3D is that game resources and objects can be imported or exported in the form of a package, which can easily make different game projects share development works. Therefore, using package can greatly improve efficiency in game development. In addition to resource material files, specific functions can be packaged, such as AI, network operation, character control, etc.

II. UNITY3D GAME ENGINE INTERNAL ANALYSIS

A. The Component Model

The component model is applied in the Unity3D game development, which provides a scalable programming architecture. Game function modules can be reused conveniently in this component model. Each entity in a game scene is called as a GameObject. A GameObject represents a game object, which has the characteristics of a container. According to the needs of a game, many different components are added into a GameObject. A component can be taken as a collection with a group of related functions, which can be accessed through an interface. For example, a script is a component and its role is to offer a logical operation on a game object, and the Box Collider component in Unity3D specifically provides a support for game objects collision detection. Unity3D has many predefined components. Programmers can combine some of them to create a feature-rich GameObject. The Figure 1 shows the ideas of the component model in Unity3D.

The Figure 2 shows the tree hierarchy for organizing objects in Unity3D game project. A game is composed of one or more scenes, each scene includes one or more GameObjects, and moreover, every GameObject is composed of some components or child GameObjects.

In game development, besides directly using GameObjects predefined in Unity3D, programmers can create an empty GameObject with the information about position, rotation and scale of an object, and then add scripts or other components

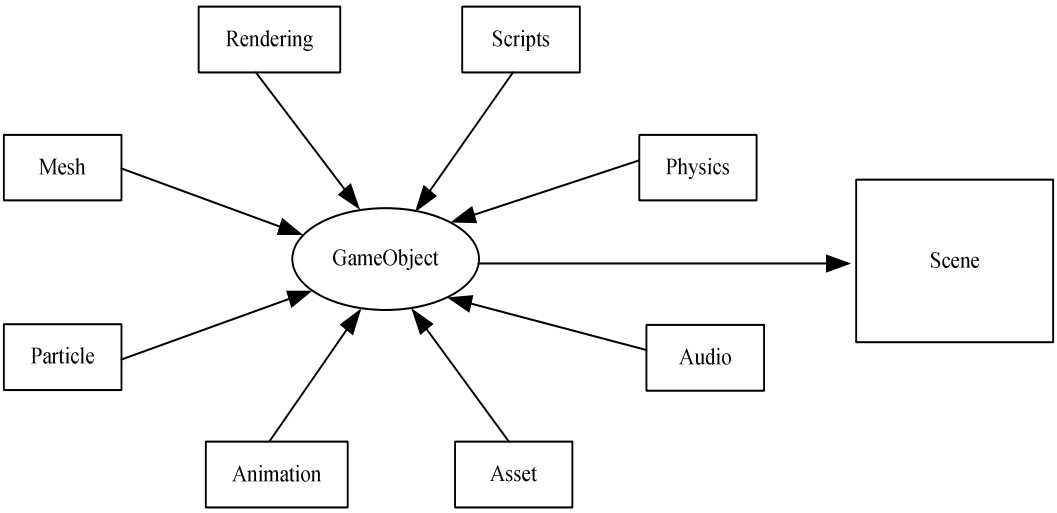

Figure 1 The Unity3D component model

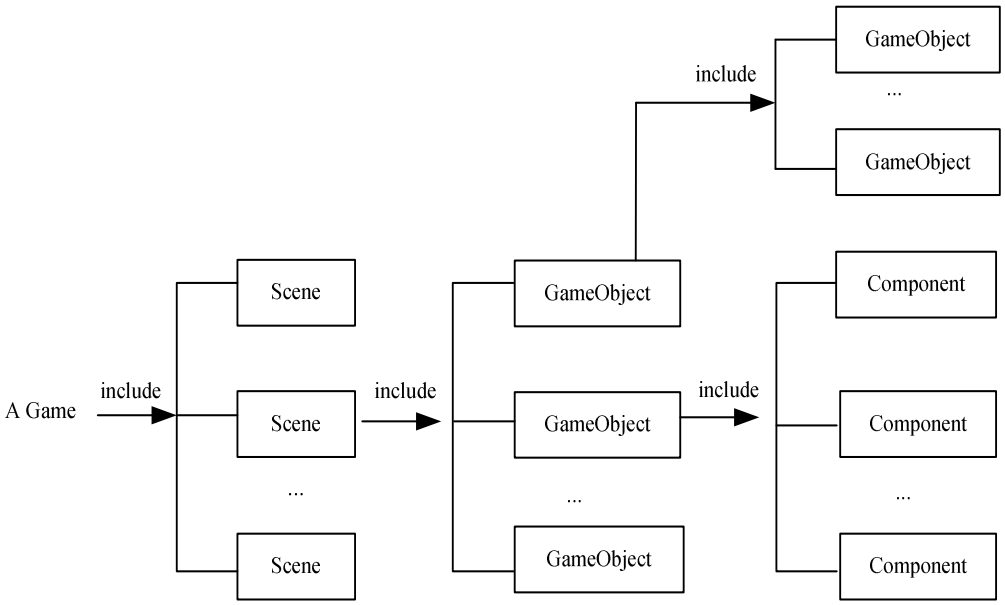

Figure 2 The Unity3D game project hierarchy

into it. In order to facilitate the same type of game object management, Unity3D affords the Prefab that is a technology like a template. A Prefab can contain both objects and game resources such as 3D models. When the same type of game objects need to be created, a Prefab can be used in this situation. All GameObjects will be updated simultaneously when its Prefab is changed. The above mechanism of Prefab can greatly improve the maintenance efficiency of a game.

To observe the impact of an object's state on a game, programmers can dynamically change the configuration parameters of a component when the game is running. After the game exits, all its initial parameters will be reset. In order to better grasp the impact of game elements, details of differences between game design and actual running affect can be discovered by using the game view and scene view at the same time.

B. The Event-Driven Programming

There are a number of scripts in a game. A script is a class controlling the behavior of a game, which should inherit a base class called MonoBehaviour. MonoBehaviour defines common event triggering methods. When a predefined event occurs, an appropriate method will be automatically executed. The Figure 3 lists the important event methods in Unity3D and their execution order relationships.

The Awake() method is executed only once in the life cycle of a script instance, when the script is called. The Awake() method can be used to initialize variables when a game starts. After all GameObjects are initialized, this method will be executed. The execution order of each Awake() method in GameObjects is random. Consequently, it is safe for referencing other GameObjects in the Awake() method without causing a null object reference.

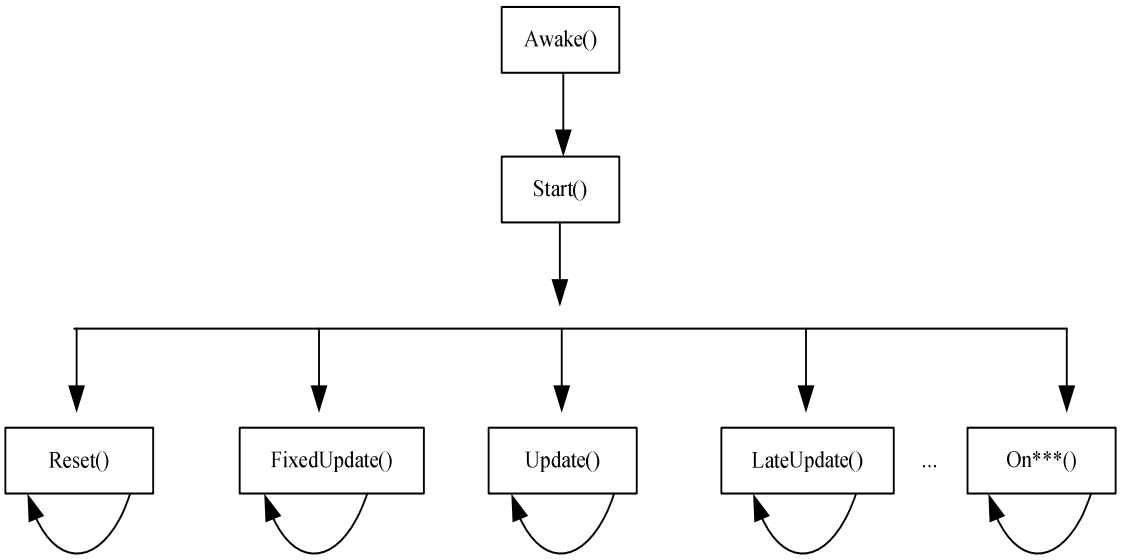

Figure 3 The event-driven model

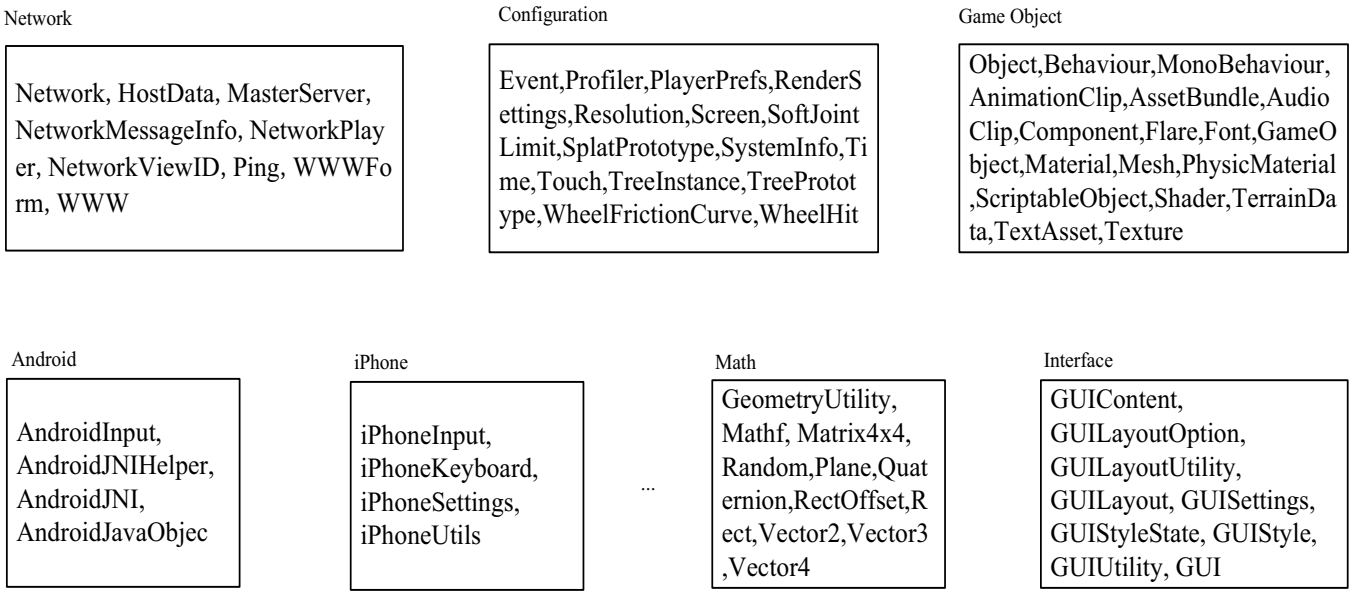

Figure 4 The distribution of main classes in Unity3D

The Start() method is executed only once like the Awake() method, and it can also be used to initialize variables. The difference between the two methods is that the Start() method is executed only when the script instance is enabled. When it needs to order initialization operations, the Start() method can be utilized to delay initializing codes.

After the execution of the Awake() and Start() methods, other event methods can be recycled to trigger in the life cycle of a game. Methods such as Reset(), FixedUpdate(), Update(), LateUpdat() can be executed in multiple times. For example, the Update() method will be called when each frame is rendered. Its implementation number depends on frames per second. It is recommended that logic operations should be put in this method.

C. The Summary of Classes

A game is controlled by programming scripts to access the internal of Unity3D. There are many system classes in Unity3D, which are useful in scripts. There are two kinds of classes. One is named as runtime classes that are applied to operate game objects, another is named as editor classes that are applied to modify Unity3D tools such as plug-ins and view information.

Although a class in Unity3D has small number of methods, it is not only easy to use and powerful to develop games. Understanding methods function is the key of using them in a game program.

Some classes represent visual game objects, which all inherit from the Object class. Ojbects from these classes can be directly showed in a game scene and players will see their effect. These objects can be created, updated and destroyed in game codes. These classes relate to the camera, sound, animation, particles, rigid, material, texture, color, font, GUI text, GUI image, light, sky boxes, etc.

Other classes are used as auxiliary operations, such as network processing, game configuration, math geometry, resource management and special classes for mobile platforms.

According to function of classes, nearly 200 runtime classes are categorized into the Figure 4.

III. THE RESEARCH ON KEY TECHNOLOGIES

A. A Dynamic Algorithm of Randomly Generating NPCs

NPC is the abbreviation of non-player character. That the number and location of NPCs are dynamically changed can keep players strong interesting on a game and extend the vitality of the game. NPCs can be designed by the difficulty and logic of a game. The following will propose a dynamic algorithm of randomly generating NPCs, which can be called when some NPCs need to be created in a game.

1. Select a kind of NPC that has not been yet handled, it is named as *t*;

2. *bCreate*=0;

3. Determine whether the kind of NPC *t* should appear:

If(*random.value*>*tRValue* && *NPCrule*==1){

//determine by a probabilistic method, *tRValue* is the appearing probability for the kind of NPC *t*.

bCreate=1;

}else if(*existNum*<*tEValue* && *NPCrule*==2){//Existing number of the kind of NPC *t* is below a certain threshold value *tEValue*.

bCreate=2;

}else if(*somethinghappen* && *NPCrule*==3){

//determine by an event in which some conditions are satisfied.

bCreate=3;

}

4. If (*bCreate*>0)

tNum=Random.Range(*tMinValue*, *tMaxValue*);//determine how many NPCs of the kind of *t* will be created.

5. while(*tNum*>0){

6. Set the location of a NPC of type *t*:

position=*reference.transform.position*+*Vector3*(*x,y,z*);//*x, y* and *z* are randomly generated;

7. Copy a NPC of type *t* from the template by using the instantiate method and set the behavior of its action;

8. *tNum*--;}

9. Determine whether it is necessary to create other kinds of NPCs, if it is necessary then goto step 1, else end this algorithm.

Now, let's analyze the algorithm. The random operations are applied in the step 3, 4 and 6 to control the number and location of new NPCs. The step 3 determines whether certain types of NPCs should be generated by a variety of rules. In the step 4, difficulty of a game can be adjusted by regulating *tMinValue* and *tMaxValue*. The numbers of NPCs are dynamic by the corresponding kind of NPC. In the step 6, the protagonist or the pre-set coordinates in a scene can be taken as a reference point. An empty GameObject with special deployment mark can be designed as a reference for the location of NPCs.

It often requires creating a number of NPCs in a game. We introduce a simple solution of enabling them to have perfect distribution.

1. Manually deploying a number of referencing coordinates for NPCs in the scene by special strategies. The distribution of references should have a well density.

2. Classifying these references by setting the tag properties in Unity 3D to prepare basic data for deployment. Programmers can quickly find a group of GameObjects with a certain tag by the *FindGameObjectsWithTag* method.

3. Deploying NPCs by these references and deployment strategies.

B. A Shooting Algorithm

There are mainly two ways of firing bullets in game development. One is that a true 3D object is modeled by decorating a bullet with texture and color. This approach needs to control the flight path of a bullet and destroy the bullet when it is out of bounds. Another method is to use ray tracing technology. That the light collides with an object means a bullet hits the object. In order to get more realistic effects, cartridge cases are pop-upped or sparks are twinkled at the gunpoint when bullets are fired. Moreover, drawing light between the launch point and target point can more realistically simulate the firing of bullets.

In a 3D shooting game, a sight is usually provided to players. Players can move a mouse to control it for firing. The principle of this technique is to place a graphic sight in the center of the screen. Moving the mouse is equal to move the camera, which will cause the illusion of the sight moving in the screen. In fact, the coordinates of the camera are real target coordinates in game codes.

In a first-person game, a main camera is set as a sub-object of the protagonist [7], so they can move together as a whole. There are four directions in the mouse movement, namely left, right, up and down movement. The following will discuss how to calculate them.

The mouse moving left or right is equivalent that the camera rotates the Y axis. The following formula is about the way of calculating a 3*3 rotation matrix $Ry(\theta)$. θ is the horizontal movement angle of the mouse, which can be got through the magnitude and direction of mouse horizontal movement. *Input.GetAxis*("*Mouse X*") method in Unity3D can help programmers do that in a simple way. This method will return a value between -1 to 1. The *sensitivityX* represents the sensitivity value for the mouse, which is usually assigned to 15. To the formula (1-3), the *Rotate* method in Unity3D can be directly applied to rotate an angle.

$$mouseX = Input.GetAxis("MouseX"); \quad (1-1)$$

$$\theta = mouseX * sensitivityX; \quad (1-2)$$

$$Ry(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad (1-3)$$

Finally, we discuss the calculation principle about moving the mouse up or down. A camera moving up or down is equivalent that the head of a person (parent node) rises up or down. Moving values are in a fixed angular range. For example, angles are between -60 degrees to 60 degrees. From the mathematical point of view, it relates to the rotation of the parent's transformation and actually it can be expressed by Euler angles [8]. Euler angles represent three angles around the *X*, *Y*, *Z* axis rotation, which are used to simulate the rotation of local coordinates. The advantage of applying Euler angles is more intuitive, easy to understand and control than a matrix. The initial value of *rotationY* in formula (2-1) is 0, later it cumulates the magnitude for mouse moving up and down. The *p* in formula (2-3) is the Euler angle rotating around the *X* axis. New Euler angles can be assigned to *transform.localEulerAngles* in Unity3D.

$$rotationY += Input.GetAxis("MouseY") * sensitivity; \quad (2-1)$$

$$rotationY = Mathf.Clamp(rotationY, minnumY, maxnumY); \quad (2-2)$$

$$p = -rotationY \quad (2-3)$$

REFERENCES

[1] Quake. http://www.idsoftware.com/games/quake/.

[2] Unreal Tournament. http://www.unrealtournament.com.

[3] Source. http://source.valvesoftware.com/.

[4] BigWorld. http://www.bigworldtech.com/index/index.php

[5] CryENGINE . http://www.crytek.com/cryengine.

[6] Unity. http://unity3d.com.

[7] Graham McAllister. Creating a First Person Shooter (FPS). http://download.unity3d.com/support/resources /files/FPS_Tutorial_1.pdf.

[8] Fletcher Dunn, lan Parberry, 3D Math Primer for Graphics and Game Development. Tsinghua University Press, 2005.