Development of Mobile Games Based on Cocos2d-x

Lingling Zhao[a], Dandan Li[b] and Xiufeng Shao[c]
Software Engineering, Beijing city University, Beijing, 100083, China
[a]zhaolingling@bcu.edu.cn, [b]ldd2003@bcu.edu.cn, [c]shaoxiufeng@bcu.edu.cn

Keywords: Cocos2d-x; Lua; Mobile Games; Shooting game

Abstract. This paper systematically discusses the technical basis, core technology, game example of Cocos2d-x. The development of mobile games based on cocos2d-x is described in detail including the building of the game development environment and the development of Lua shooting game.

Introduction

With the advent of the information technology, game products and services have sprung up and played a role that cannot be ignored in enriching lives and providing quick information. Cocos2d is an open source framework based on the MIT protocol for building the interactive applications of games, programs and other graphical interfaces.

How to Build the Game Development Environment

Installation of the Programming Language Lua. The plug-in lua should be installed first in the Eclipse development environment – open Eclipse, help-> install new software, and then click add, enter http://download.eclipse.org/ldt/releases/milestones/ in the location input box, and then click OK, as shown in Figure 1.

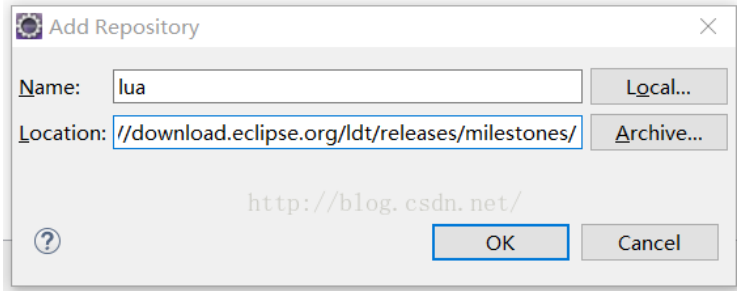

Figure 1 Install plug-in lua

Click ok, the plug-in lua will be shown below, select for installation. Eclipse can be used to develop the lua script after the installation. The operating environment after the installation is shown in Figure 2.

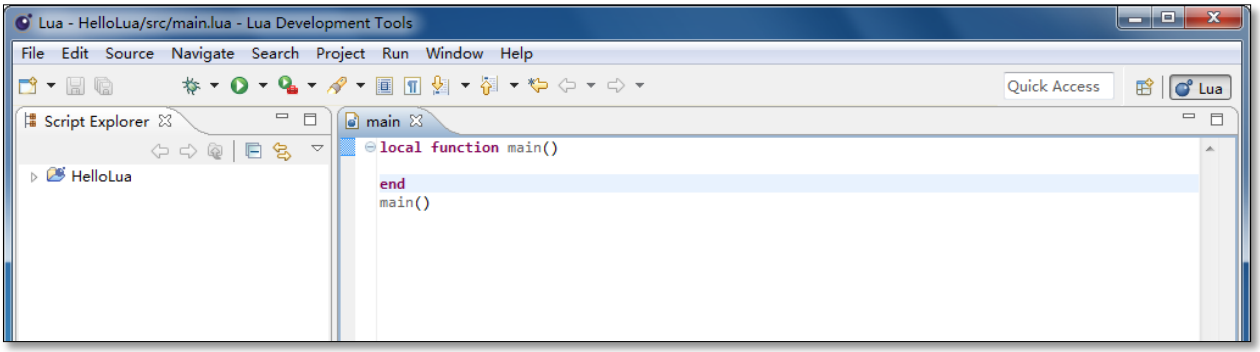

Figure 2 Lua Operating Environment

Cocos2d-x Lua Environment Building. Cocos Code IDE is used to develop the Cocos2d-x game for lua. The tool can be installed and used in a very simple way. Cocos Code IDE can be

downloaded at http://www.cocos2d-x.org/download. Select the appropriate file, which currently includes the Mac OS X version and Windows version, and note there are 32-bit and 64-bit Windows.

Run Cocos Code IDE, and select the directory Workspace during the start-up process. The directory is for project management, where the game projects later created will be saved by default. If the directory does not exist, the system will recreate it.

Introduction to Hello Cocos2d-x Lua

Creation of default game projects. (1) Open Cocos Code IDE and create the new Lua game project.

(2) Click “next step”, and then complete the building of the game project. Select the HelloLua project in the project navigation panel on the left, and right-click the menu to select Run As → Cocosluabinding to run the game, as shown in Figure 3.

Figure 3 Game Running Result

(3) The game project created has been able to run up. Then an introduction will be made to the file structure in the HelloLuaGame project. Use Cocos Code IDE to open the game project navigation panel on the left, as shown in Figure 4.

- The resource files of the games are stored in the res folder, such as picture resources, sound resources and font resources.
- The source code is stored in the src folder, and the current default files are main.lua and GameScene.lua.
- main.lua is the file for program entry, and Cocos2d-x will bind the file at the bottom and start running it.
- GameScene.lua is the file for the game scene, and the other game codes will be mainly implemented here.

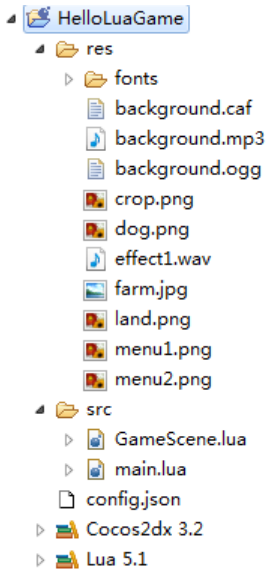

Figure 4 Navigation Panel

Recreation of HelloLuaGame. (1) The default created game is somewhat complicated for beginners. Now I will complete a simple task of game scene development. First a new lua file will be created in the src folder, named HelloWorld.

(2) Open the file HelloWorld.lua, first import the file Lua common in Cocos2d-x engine, including Cocos2d and Cocos2dConstants:

```
require "Cocos2d"
require "Cocos2dConstants"
```

(3) Define a scene for HelloWorld, and a create function and createLayer function of one class, and finally return to the scene object, with the code as follows:

```
require "Cocos2d"
require "Cocos2dConstants"
local HelloWorld=class("HelloWorld",function()
        return cc.Scene:create()
        end)
function HelloWorld.create()
        local scene=HelloWorld.new()
        scene.addChild(scene:createLayer())
        return scene
end
function HelloWorld:createLayer()
end
return HelloWorld
```

(4) There is no layer in the current HelloWorld scene, we need to create a layer in the createLayer function, with the function to display a Hello World, and the code is as follows:

```
function HelloWorld:createLayer()
        local layer=cc.Layer:create()
        local label=cc.LabelTTF:create("Hello World","Arial",46)
        size=cc.Director:getInstance():getWinSize()
        label:SetPosition(cc.p(size.width/2,size.heigth-label:getContentSize().height))
        layer:addChild(label)
        return layer
end
```

(5) After the above operation is completed, we only need to load HelloWorld scene in main.lua, and also delete the playback event of the default background music:

```
--create scene
local scene = require("GameScene")
local gameScene = scene.create()
gameScene:playBgMusic()
```

(6) See Figure 5 for the final operation effect:

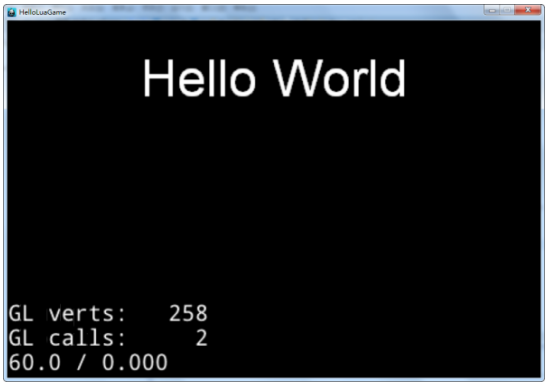

Figure 5 Game Running Result

Example of the Shooting Game Development based on Lua

It is a shooting game revolving around an old bomber in World War II. It flies across the universe, time and space after getting lost. It fights against the enemies while avoiding the creatures and asteroids in the virtual space. The game is created in the following steps:

Creation and Preparation of the Game Project. Cocos Code IDE is used to create a LostRoute game project. Prepare the resource files for the game including:

- Font File: hanyi.ttf
- Tile map: blue_bg.tmx and red_bg.tmx
- Texture Atlas: loading_texture.plist and LostRoutes_Texture.plist
- Particles: explosion.plist, fire.plist, light.plist
- Sound effects: Blip.wav, Explosion.wav, game_bg.mp3 and home_bg.mp3

Creation of the Loading Scene. The Loading scene is the first interface one sees in the game, where resources for texture cache and preprocessed sound are loaded. First add the Loading scene and layer, in which the text “Loading” is animated, so as to shorten the psychological time of waiting, as shown in Figure 6.

Creation of the Home Scene. The Home scene is the main menu interface, through which you can enter the game scene, Setting scene and Help scenes. Cocos Code IDE is used to create the HomeScene.lua file for the Home scene. The scene is mainly a menu containing three menu items, as shown in Figure 7.

Figure 6 Loading Scene

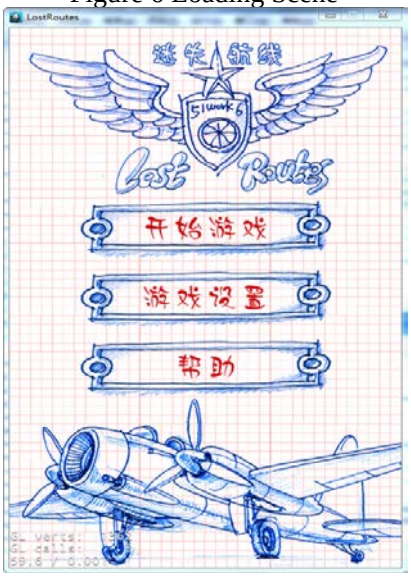

Figure 7 Home Scene

Creation of the Setting Scene. Set the scene to achieve the background music sound switch control, through the Cocos Code IDE tool to create settings scene SettingScene.lua file, the main function of the scene is a menu contains two switch menu items, as shown in Figure 8.

Creation of the Help Scene. The Help scene aims to show some of the information about the game. Cocos Code IDE tool is used to create the file HelpScene.lua for the Settings scene, as shown in Figure 9.

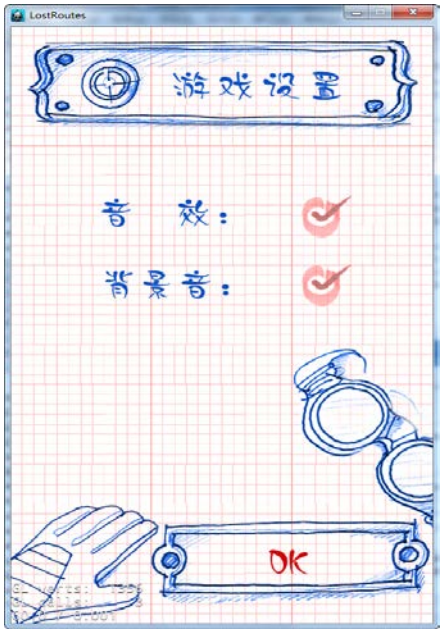

Figure 8 Setting Scene

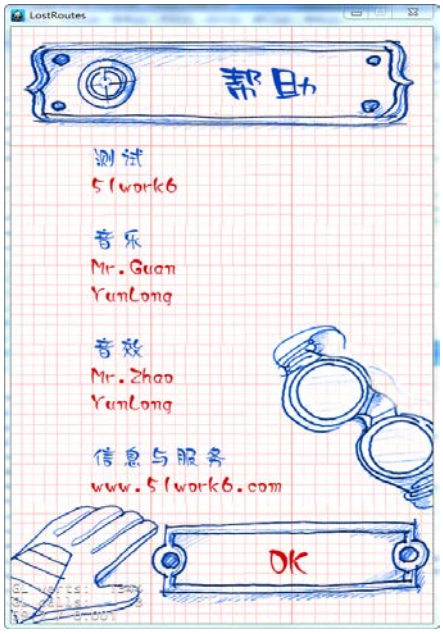

Figure 9 Helping Scene

Realization of the Creation of the Main Scene. The main task of game development is to realize the main scene, Cocos Code IDE is used to create the main scene file GamePlayScene.lua:

Creation of the Enemy Wizard. The creation of enemy wizard is complicated, to which packaging according to the need instead of cc.Sprite can be applied. An Enemy class should be created to inherit cc.Sprite, and the enemy wizard class related functions and members defined.

Creation of the player's aircraft wizard. The creation of the player's aircraft wizard is not as complex as that of the enemy wizard, for there is only one player's aircraft. The cc.Sprite class should be inherited, and the functions and members unique to the aircraft wizard class Fighter defined.

Creation of the Shells Wizard. The cc.Sprite class is also not directly applied to Bullet, which is packaged to inherit the cc.Sprite class.

The Initiation Scene. As the specific game scene is complex, we will realize the game scene in several steps. First let's look at the initialization of the game scene. In this part, functions ctor, createInitBGLayer, onExit and onEnter are involved. The functions ctor and createInitBGLayer can be considered at once, because they are called when the scene is initialized and only once in the scene lifecycle. The infrequently changing sprites and the backgrounds can be implemented here. The onExit function mainly cleans up the resources, including stopping the game scheduling, canceling the time listener and deleting the nodes. The onEnter function mainly calls the createLayer function to create the game main layer.

Realization of the Game Scene Menu. There are three menus in the game scene: pause, return to home page and resume game. The pause menu is located in the upper left corner of the scene. Clicking the pause menu will bring up the "return to home page" and "resume game" menu; therefore, one needs to initialize the pause menu in the OnEnter function.

Player's Aircraft Firing Shells. Player's aircraft needs to continue firing shells, which does not require the control by the player. One only needs a scheduling plan for repeated firing at a certain time. Start the game scheduling in the functions OnEnter and menuResumeCallback.

Detection of Collision between Shells and the Enemy. The game needs to detect the collision between shells fired by the player and the enemy and that between the player's aircraft and the

enemy. In order to detect collisions between objects, it is necessary to register the contact listener event in the function onEnter. When the shells collide with the enemy, one way will be used to deal with the corresponding follow-up events.

Detection of Collision between Player's Aircraft and the Enemy. The detection of collision between player's aircraft and the enemy is similar to that between the shells and the enemy, which is achieved in the contact listener event.

Display of the Health Point of Player's Aircraft and Player's Score. The update of the health point of player's aircraft is needed when it changes. When the player's aircraft collides with the enemy or the player scores more than 1,000 points each time, the health point needs update by calling the function updateStatusBarFighter; the update of the player's score is displayed after the elimination of an enemy by calling the function updateStatusBarFighter.

Realization of the creation of the End Scene. One can enter the End scene of the game through the Game scene after the game is over. We need to show the highest records in the End scene, which are saved in UserDefault. When the End scene is realized, the score from the previous scene should be received.

References

[1] Shen Dahai. Explanation for Cocos2d-x based Mobile Game Development and Related Projects. Beijing: Tsinghua University Press. 2014

[2] Wang Yongbao. Mastery of Cocos2d-x based Game Development (Fundamental). Beijing: Tsinghua University Press. 2016

[3] Ou Tongtong. Detailed Examples of Cocos2d-x based Game Development. Beijing: Tsinghua University Press. 2016

[4] David Young. Lua Game AI Development Guide. Beijing: Posts & Telecom Press. 2017

[5] Guan Dongsheng. Cocos2d-x Examples: Lua (second edition) Guan Dongsheng. Beijing: Tsinghua University Press. 2017

[6] Zhao Zhirong, Guan Dongsheng. Study Notes on Cocos2d-x – Mastery of Lua API and Game Project Development. Beijing: Tsinghua University Press. 2016