****

****

**本 科 毕 业 设 计**

**院 系** 软件学院

**专 业** 软件工程

**题 目** **基于Cocos2D的HTML5游戏的设计与实现**

**年 级** 2012级 **学 号** 121250119

**学生姓名** 沈剑伟

**指导教师** 郑滔 **职 称** 教授

**论文提交日期** 2016/05/04

**南京大学本科生毕业论文（设计）中文摘要**

毕业论文题目： 基于Cocos2D的HTML5游戏的设计与实现

软件学院 院系 软件工程 专业 2012 级本科生姓名： 沈剑伟

指导教师（姓名、职称）： 郑滔 教授

摘要：

随着2012年12月17日，W3C宣布HTML5规范正式定稿，移动互联网时代迎来了高速发展的时间。手机应用、Web应用、Hybrid应用如雨后春笋般涌入了用户的眼中，而其中游戏行业的增长尤为引人注目。除了作为一种盈利产品以外，游戏作为一种生动有趣并且能够充分和用户交互的运营策略也随着HTML5的到来，逐步登堂入室，成为广大运营人员的常用手段。

Cocos2D是一个基于MIT协议的开源框架，用以构建不同平台的游戏，能够大量节省开发者的开发时间。而Cocos2D-html5是基于HTML5规范集的Cocos2D引擎分支，具有强大的跨平台能力和性能，并且联合Cocos2D-X Javascript-binding可以最大程度地实现游戏的跨平台运行。

本文以心动网络股份有限公司平台事业部《快斩狂刀》HTML5推广游戏为例，重点阐述了基于Cocos2D的HTML5游戏实现和跨平台游戏打包的实践：如何在不同尺寸屏幕的移动终端上创建始终适配的游戏，并且打包部署到现有项目环境中。此推广游戏是一个考验反应的横屏防守游戏，通过玩家点击屏幕在正确的时机来格挡敌人的攻击以触发主角防守反击，记录玩家最后防守成功次数并根据成绩发放《快斩狂刀》游戏内奖励，调用用户微信信息生成排行榜。我在项目主要担任游戏的设计编写和打包部署。

关键词：Cocos2D；HTML5；游戏；

**南京大学本科生毕业论文（设计）英文摘要**

THESIS：Implementation of an HTML5 Game Based on Cocos2D

DEPARTMENT： Software Institute

SPECIALIZATION: Software Engineering

UNDERGRADUATE: 2012

MENTOR: Tao Zheng

ABSTRACT：

With the formally finalizing of HTML5 specification by W3C on 17th of December in 2012, the Internet in the field of mobile device has welcomed a prosperous age since then. Mobile applications ,web applications and Hybrid applications has been flooding into the market, among which mobile games are outstanding. Excepting for playing the role of profited product, mobile game has also become a kind of lively and interesting strategy which could fully interact with users with the coming of HTML5, and it has been appreciated by operational staff.

Cocos2D is an open source framework which could be used to build cross-platform games which would save lots of time of developers. And Cocos2D-html5 is a branch of the engine which is based on the HTML5 Specification, having the great ability of crossing platform and performing well. With Cocos2D-X javascript-binding, it could make games run on different platforms quite well.

This thesis gives an example of a game used to promoting the game ‘Kuai Zhan Kuang Dao’, which is made by Platform Department of XinDong Shanghai Ltd, focusing on the building process of an HTML5 game based on Cocos2D and the practice of packaging for different native platforms. In the working process, we tried to make our game adapt to different mobiles with all kinds of screen and we finally deployed it to our current environment. This game is a kind of defending game which tests the responding of users by tapping the screen at the right moment to defend the attack of enemies which finally triggers the counterstrike of protagonist the users are playing , and it will recording the number of eliminated enemies and give out rewards to gamers of ‘Kuai Zhan Kuang Dao’ .Users can also see their position in a rank list which using their Wechat information. My duty is all about the design of the game and coding issues with packaging and deploying.

KEYWORDS: Cocos2D;HTML5;Game

**目 录**

目录 I

[图目录 IV](#_Toc450421575)

[表目录 V](#_Toc450421576)

[第一章 引言 1](#_Toc450421577)

[1.1 项目背景 1](#_Toc450421578)

[1.2 项目目标 2](#_Toc450421579)

[1.3 现有游戏引擎分析与比较 2](#_Toc450421580)

[1.3.1 Construct2 2](#_Toc450421581)

[1.3.3 DOM 3](#_Toc450421582)

[1.4 本文工作 3](#_Toc450421583)

[1.5 本文结构 3](#_Toc450421584)

[第二章 技术概述 5](#_Toc450421585)

[2.1 Cocos2D 5](#_Toc450421586)

[2.1.1 SpriteSheet&SpriteFrame 5](#_Toc450421587)

[2.1.2 缓存池 5](#_Toc450421587)

[2.1.3 BakeLayer 5](#_Toc450421589)

[2.1.4 ParticleSystem 6](#_Toc450421590)

[2.1.5 渲染控制 6](#_Toc450421591)

[2.1.6 OpenGL 7](#_Toc450421592)

[2.1.7 混合模式 7](#_Toc450421593)

[2.2 Jasmine 7](#_Toc450421594)

[2.3 CakePHP 8](#_Toc450421595)

[第三章 快斩格挡需求分析与概要设计 10](#_Toc450421597)

[3.1 系统整体概述 10](#_Toc450421598)

[3.1.1 系统功能 10](#_Toc450421599)

[3.2 系统的需求分析 10](#_Toc450421600)

[3.2.1 用例描述 10](#_Toc450421601)

[3.2.2 用例细化 13](#_Toc450421602)

[3.3 系统的概要设计 22](#_Toc450421603)

[3.3.1 系统的模块划分 22](#_Toc450421604)

[3.3.2逻辑视图与开发视图 23](#_Toc450421606)

[3.3.2 逻辑视图与开发视图的映射 24](#_Toc450421607)

[3.4 本章小结 25](#_Toc450421608)

[第四章 快斩格挡的详细设计与实现 26](#_Toc450421609)

[4.1 模块概述 26](#_Toc450421610)

[4.2游戏控制模块和战斗模块的详细设计 26](#_Toc450421611)

[4.2.1 游戏控制模块的详细设计 26](#_Toc450421612)

[4.2.1.1 承担的需求 26](#_Toc450421613)

[4.2.1.2 游戏控制模块类图 27](#_Toc450421614)

[4.2.1.3 游戏控制模块的相关类描述 27](#_Toc450421615)

[4.2.1.4 游戏控制模块的顺序图 30](#_Toc450421616)

[4.2.2 战斗模块的详细设计 31](#_Toc450421617)

[4.2.2.1 承担的需求 31](#_Toc450421618)

[4.2.2.2 战斗模块的类图 32](#_Toc450421619)

[4.2.2.3 战斗模块的相关类描述 32](#_Toc450421620)

[4.2.2.4 战斗模块的顺序图 39](#_Toc450421621)

[4.3 游戏控制模块和战斗模块的实现 41](#_Toc450421622)

[4.3.1 游戏暂停、恢复 41](#_Toc450421623)

[4.3.2 生成敌人 42](#_Toc450421623)

[4.3.3 防守 45](#_Toc450421625)

[4.3.4 敌人攻击 48](#_Toc450421626)

[4.4 本章小结 51](#_Toc450421627)

[第五章 总结与展望 52](#_Toc450421628)

[5.1 总结 52](#_Toc450421629)

[5.2 展望 52](#_Toc450421630)

[参考文献 53](#_Toc450421631)

[致谢 54](#_Toc450421632)

# 图目录

[图1.1移动游戏用户数量逐年柱状图 1](#_Toc450435078)

[图1.2 HTML5游戏用户数量柱状图 2](#_Toc450435079)

[图2.1 绘制管线 6](#_Toc450435080)

[图2.2 Jasmine 8](#_Toc450435081)

[图2.3 cakephp 9](#_Toc450435082)

[图3.3 防守状态图 16](#_Toc450435083)

[图3.4 防守顺序图 17](#_Toc450435084)

[图3.5 系统模块划分 22](#_Toc450435085)

[图3.6 逻辑视图 23](#_Toc450435086)

[图3.7 开发视图 24](#_Toc450435087)

[图4.1 游戏过程控制类图 27](#_Toc450435088)

[图4.2 开始游戏顺序图 30](#_Toc450435089)

[图4.3 暂停游戏顺序图 31](#_Toc450435090)

[图4.4 战斗模块类图 32](#_Toc450435091)

[图4.5 英雄顺序图 39](#_Toc450435092)

[图4.6 生成敌人顺序图 40](#_Toc450435093)

[图4.7 敌人攻击顺序图 41](#_Toc450435094)

[图4.8 GameScene类代码 42](#_Toc450435095)

[图4.9 EnemyController类代码 43](#_Toc450435096)

[图4.9 GameService类代码 44](#_Toc450435097)

[图4.10 FightLayerUI类代码 45](#_Toc450435098)

[图4.11 onDefend方法实现 46](#_Toc450435099)

[图4.12 HeroService.defend方法实现图 47](#_Toc450435100)

[图4.13 HeroSprite.defend的具体实现 48](#_Toc450435101)

[图4.14 HeroController.idle方法实现 49](#_Toc450435102)

[图4.15 EnemySprite类代码 50](#_Toc450435103)

[图4.16 HeroSprite.getHurt具体实现 51](#_Toc450435104)

# 表目录

# 第一章 引言

## 1.1 项目背景

随着2012年12月17日，W3C宣布HTML5规范正式定稿，移动互联网时代迎来了高速发展的时间。手机应用、Web应用、Hybrid应用如雨后春笋般涌入了用户的眼中，而其中游戏行业的增长尤为引人注目。根据DataEye在15年第二季度提供的数据，移动游戏用户规模在12年到15年间实现了爆发式的增长。

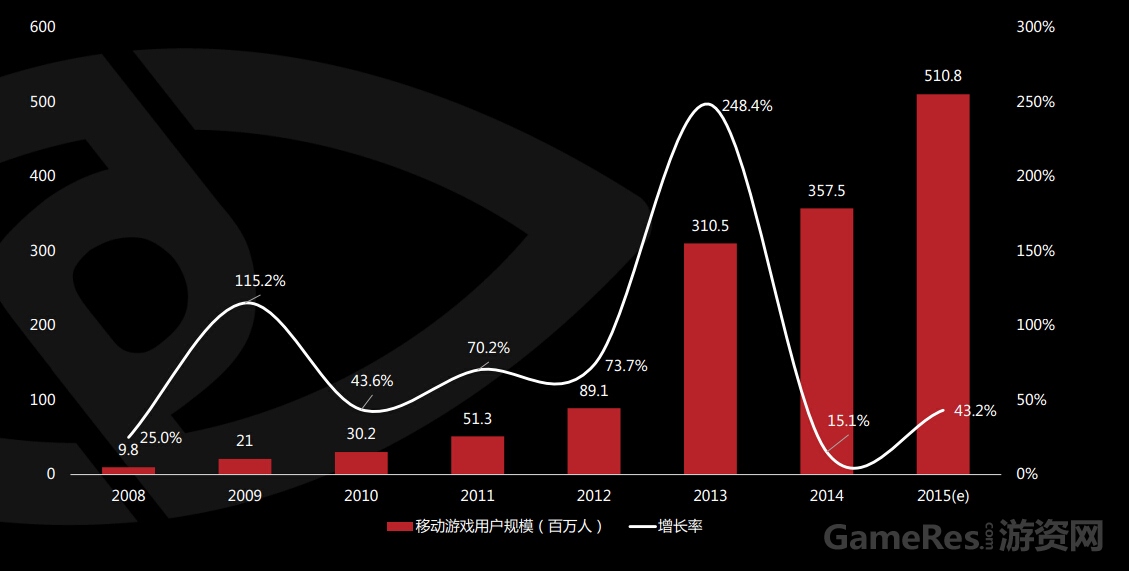


图1.1移动游戏用户数量逐年柱状图

移动游戏用户规模的迅速增大以及各大浏览器厂商对HTML5 Canvas和其他特性的不断支持给了HTML5游戏以无限的可能。同样来自DataEye的数据显示，今年来HTML5游戏用户规模正在逐渐逼近原生游戏用户规模。可以说HTML5游戏是游戏行业接下去的一个大热点，在VR技术还未完全普及的近几年，可以预测HTML5游戏将保持强劲的生命力。

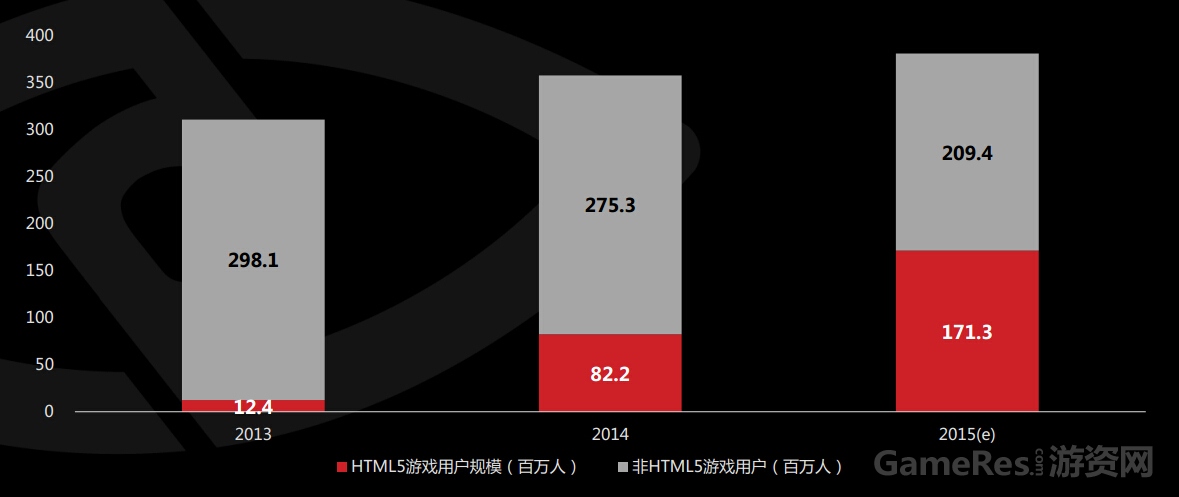


图1.2 HTML5游戏用户数量柱状图

HTML5游戏的开发有着原生应用开发不具备的低门槛、短周期、跨平台、传播快等优点，虽然性能缺点也十分明显，但是随着硬件技术的不断进步，这样的差距将会保持在一个可以接受和权衡的范围内。

众所周知，互联网公司前期的用户积累和流量积累至关重要，而这就需要好的营销和推广策略，在微信的崛起和称霸的年代，很少有公司会绕过这个巨大体量的应用去推广自己的产品。而其中一种推广策略就是简单的HTML5小游戏，当然作为游戏公司，在预热即将推出的游戏时，先放出一个简单的HTML5小游戏也可以达到宣传和提前积累用户的目的。出于这个想法，游戏运营人员希望平台事业部能够快速开发一个HTML5游戏来宣传《快斩狂刀》。

## 1.2 项目目标

本游戏作为心动网络《快斩狂刀》的预热游戏，旨在通过微信公众号推广《快斩狂刀》的知名度积累用户量。

## 1.3 现有游戏引擎分析与比较

## 1.3.1 Construct2

Construct2是一款跨平台二维游戏开发引擎，不需要编码，通过定义各个部件和事件完成html5的游戏开发。该引擎可以将开发的游戏封装成多种形式，如phonegap/cocoonjs，再用相应的工具生成iOS和Android的应用。它最大的特点是不需要编程，简单直观，入门容易，在开发射击游戏上比较方便。但是弱点也比较突出，工具本身无法打包应用，必须借助第三方工具。

## 1.3.3 DOM

这里所说的DOM并不是一个引擎，而是指不使用引擎，直接用原生的HTML5进行开发。优点是规避了引擎可能带来的设计缺陷，降低了学习成本，而缺点是提高了设计的成本，不利于开发比较大型的游戏，开发比较耗时。

## 1.4 本文工作

本文介绍了基于Cocos2D-js实现的防守游戏的相关理论知识，游戏开发的流程以及相关技术，在实践项目中，我负责了游戏前端展示的所有功能的设计和编写，在本文中将着重阐述，后端API将比较简略的叙述。

## 1.5 本文结构

第一章：概述和前言部分，主要介绍了项目背景，对比分析了现有技术的优缺点，并描述了本文的结构和文章主要内容。

第二章：主要介绍在游戏开发过程中使用到的相关技术。

第三章：介绍了项目进行需求开发的过程，着重叙述需求分析的部分，从不同角度分析该游戏的需求并描述概要设计。

第四章：从整体架构设计、模块设计、包设计以及类设计的角度阐述整个游戏的详细设计并使用UML辅助分析。

第五章：总结该项目已经实现的功能，寻求可能的突破，并反思在开发过程中遇到的问题、缺陷和不足，展望项目的发展方向。

第六章：向在开发过程中以及论文撰写过程中给予帮助的师长、同事致以谢意。

# 第二章 技术概述

## 2.1 Cocos2D

Cocos2D是由触控科技开发的一个基于MIT协议开源的游戏开发框架，目前的主流版本是Cocos2D-X，这个版本有三个主要分支，Javascript-binding/Lua Binding/C++，让开发者可以根据实际情况从JS/Lua/C++中选择开发语言。在3.0版本后原来的Cocos2D-html5与Javascript-binding合并为Cocos2D-js。使用Cocos2D，开发者可以轻松开发规范化，流程化的跨平台的2D游戏，在团队协作上，Cocos-studio使得合作变得更加轻松。

## 2.1.1 SpriteSheet&SpriteFrame

在网页开发中，有一项广为人知的技巧—css sprite，也就是将许多小图拼接成一张大图，在加载资源的时候，原来需要多次HTTP请求的情况被一次请求所替代，极大提高了网页加载速度。在Cocos2D-js中也有同样的技术。在制作SpriteSheet的时候，我们用到了TexturePacker来生产对应的.plist文件，这个文件记录了图片的信息。使用Cocos2D-Js中的spriteFrameCache可以将大图分解成一个个SpriteFrame并缓存到内存中。

### 2.1.2 缓存池

如果游戏中有大量的精灵被创建和销毁而不做缓存的话，势必对游戏的性能带来比较大的损害。Cocos2D-Js有一个现成的接口可以实现缓存池，使得我们无需自行实现，节省了开发时间

## 2.1.3 BakeLayer

在游戏中，UI界面和一部分游戏界面是静态的，但是在每一帧中却按照动态内容重绘，这就造成了渲染时间的浪费。Cocos2D-Js针对Canvas 2D环境在Layer中提供了bake方法，如果该Layer自身和子节点都保持不变，那么我们可以将Layer缓存起来，后续每帧重绘只需要将这个位图绘制到舞台上即可。

## 2.1.4 ParticleSystem

在计算机图形学中，粒子系统用于模拟一些特定的模糊现象，而这些现象用其他传统的渲染技术(例如帧动画)难以展现足够的真实感。Cocos2D-Js中提供了ParticleSystem类来模拟粒子系统。

## 2.1.5 渲染控制

游戏图像的显示都遵循同样的渲染管线，渲染管线也就是图片文件从加载到最终显示到屏幕的过程，会经过CPU到GPU的一系列模块的处理。如果游戏使用了GPU加速那么游戏中的每个节点都会被推送到GPU进行渲染。CPU中负责节点的创建控制节点的位置和形变等，这些都会作为逻辑数据被存储在内存中，在每帧逻辑完成后，游戏将所有节点推送到GPU中，由GPU进行顶点处理、装配、栅格化、纹理贴图和透明度深度排序等操作，最后显示到屏幕上。

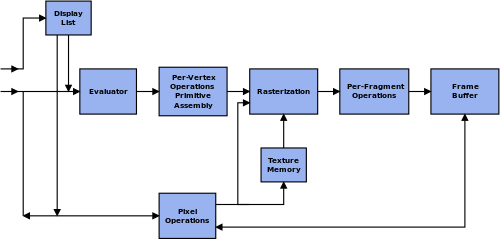


图2.1 绘制管线

## 2.1.6 OpenGL

OpenGL是定义了一个跨编程语言、跨平台的编程接口的一套规格，它也是一套专业的图形程序接口，是一个功能强大，调用方便的底层图形库。而OpenGL ES则是OpenGL三维图形API的子集。OpenGL ES 是从OpenGL裁剪而来。Cocos引擎底层使用的是OpenGL ES，编写自定义着色器的时候需要遵循OpenGL ES的规范，使用OpenGL着色语言。

## 2.1.7 混合模式

混合模式的主要功效是控制当前节点和底下的已有图像如何叠加。针对每一个像素点，OpenGL会把源颜色(当前颜色)和目标颜色(底层图像)各自取出，并乘以一个系数，最后相加得到新的颜色，用这个颜色作为像素的最终结果。混合模式是在GPU中处理的最后一步。在游戏中我们可以使用这个方法实现一些光照或者光晕的效果。

## 2.2 Jasmine

Jasmine是一个用来编写Javascript测试的框架，它不依赖于其他Javascript框架，对DOM和BOM也没有什么要求，使用起来比较简单。它所提倡的行为驱动开发使用几乎近于自然语言的方式描述了软件的行为过程，因此，可以直接作为软件的需求文档，也可以直接应用到测试中，作为测试的标准文档。我们在做单元测试时，经常是针对某个函数，或是某个类进行测试，但是被测函数或是被测的类是可能经常变化的，我们的测试案例也需要经常性的随之变化。然后，BDD描述的是软件的整个系统行为，几近于需求文档，可变性大大减小。因此，测试案例不需要做太大变化。同时，这样的测试案例最贴近于需求，贴近于实际的系统行为。  
BDD描述的行为就像一个个的故事(Story)，系统业务专家、开发者、测试人员一起合作，分析软件的需求，然后将这些需求写成一个个的故事。开发者负责填充这些故事的内容，测试者负责检验这些故事的结果。通常，会使用一个故事的模板来对故事进行描述。例如：

@Test  
public void shouldBeAbleToEditAPage() {  
    Given.thatThe(wiki).wasAbleTo(beAtThe(PointWhereItHasBeen.JUST\_INSTALLED));  
    And.thatThe(user).wasAbleTo(navigateToTheHomePage());  
  
    When.the(user).attemptsTo(changeTheContent().to("Welcome to Acceptance Test Driven Development"));  
  
    Then.the(textOnTheScreen().ofThe(user)).shouldBe("Welcome to Acceptance Test Driven Development");  
}

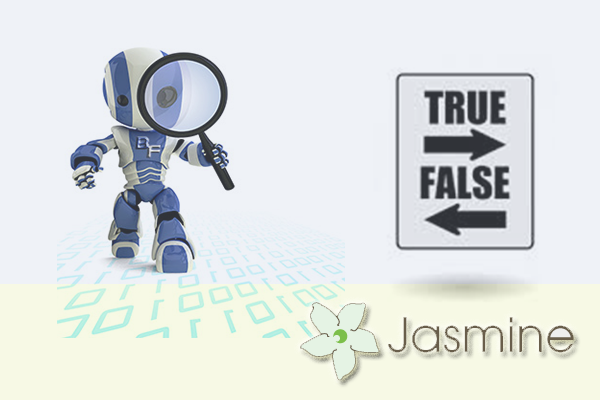


图2.2 Jasmine

## 2.3 CakePHP

CakePHP是一个运用了诸如ActiveRecord、Association Data Mapping、Front [Controller](http://baike.baidu.com/view/1805013.htm)和MVC等著名设计模式的快速开发框架。该项目主要目标是提供一个可以让各种层次的PHP开发人员快速地开发出健壮的Web应用，而又不失灵活性。

## 

图2.3 cakephp

# 第三章 快斩格挡需求分析与概要设计

## 3.1 系统整体概述

在游戏运营过程中，常常需要做一些推广活动来为即将上线的游戏造势，积累用户，游戏运营人员希望有一个简单的HTML5小游戏放置在微信公众平台上来完成上述目的。这个游戏应当是一个比较简单的横版防守游戏，只有在正确的时机点击屏幕才能触发成功的防守，并反击消灭敌人。最后记录用户消灭的敌人数发放奖励。

## 3.1.1 系统功能

每隔一段时间，从屏幕右侧会刷新敌人，当敌人走到固定位置后会开始攻击，玩家在敌人刚开始攻击的时候按屏幕任何一处，主角做出相应的防守动作视为成功防守，此时若有不止一个敌人攻击，则视攻击无效。当玩家防守成功后，主角做出反击动作，此时若有敌人攻击也视为攻击无效，反击效果将屏幕上所有敌人消灭，主角回归空闲等待状态。如此反复，随着难度不断提升，主角生命值降至0时，游戏结束，记录下玩家一共消灭的敌人数目，发送给服务器端，根据预设等级返回用户成绩和对应的奖励兑换码。

## 3.2 系统的需求分析

## 3.2.1 用例描述

表3.1 用例表

|  |  |
| --- | --- |
| UC-01 | 用户可以查看自己的排名 |
| UC-02 | 用户可以查看排行榜前十名 |
| UC-03 | 用户点击开始游戏 |
| UC-04 | 用户点击屏幕后触发主角防守 |
| UC-05 | 用户可以暂停游戏 |
| UC-06 | 用户可以恢复游戏 |
| UC-07 | 用户可以重新开始游戏 |
| UC-08 | 用户可以退出游戏 |
| UC-09 | 用户在游戏结束后获得成绩及奖励 |

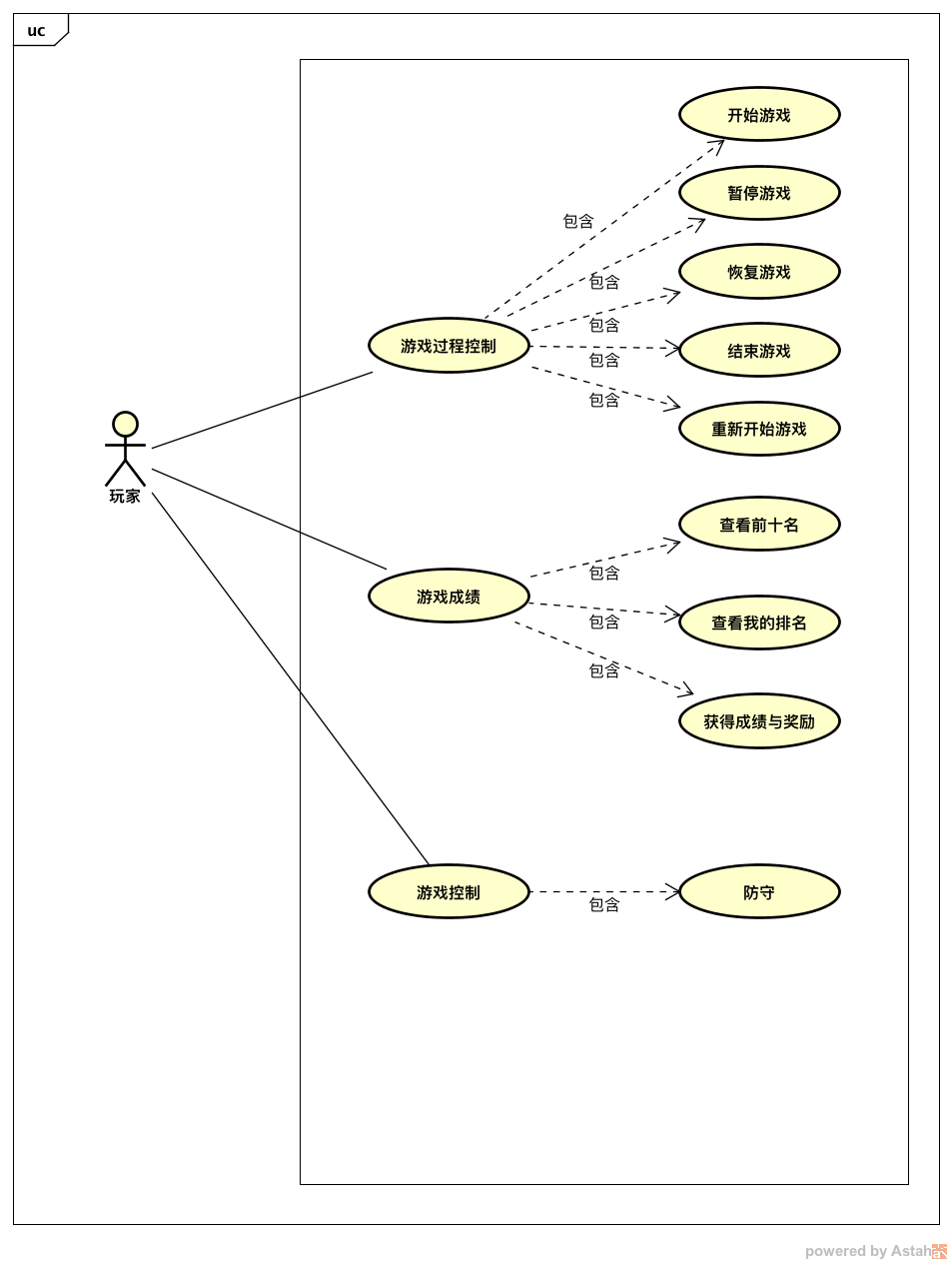


图3.1 用例图

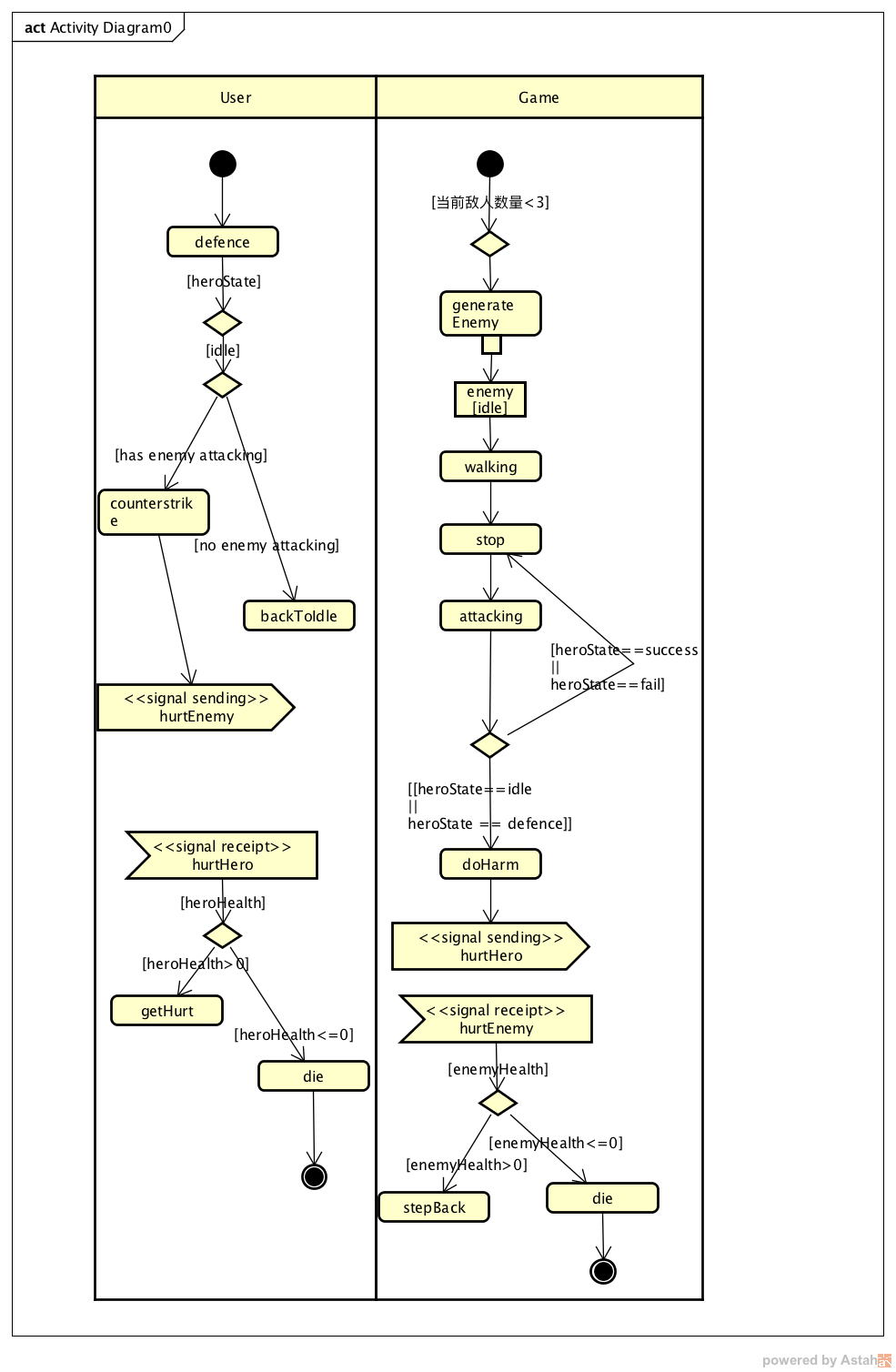


图3.2 活动图

## 3.2.2 用例细化

表3.2 查看用户排名用例表

|  |  |
| --- | --- |
| ID | UC-01 |
| 名称 | 查看用户排名 |
| 参与者 | 主参与者：游戏玩家  目的：在游戏开始前或结束后查看自己在所有用户中的排名 |
| 描述 | 产生原因：用户需要查看自己在所有用户中的排名  过程：用户点击排行榜  输出结果：系统返回用户的微信头像、排名以及成绩 |
| 优先级 | 10 |
| 触发条件 | 用户点击排行榜 |
| 前置条件 | 用户进入开始界面或处于游戏结束界面 |
| 后置条件 | 系统显示用户在所有用户中的排名、成绩以及微信头像 |
| 正常流程 | 1. 点击排行榜按钮 2. 系统显示用户微信头像、成绩和排名 |
| 分支流程 | 无 |
| 异常流程 | 2.1 网络链接异常，系统用弹窗提示用户，头像、成绩、排名使用默认值 |
| 相关用例 | 无 |

如表3.2所示，用户在游戏开始前或者游戏结束后可以查看当前最好成绩在所有用户中的排名，并接入微信接口，用微信头像来标示用户。

表3.3 查看排行榜用例表

|  |  |
| --- | --- |
| ID | UC-02 |
| 名称 | 查看排行榜 |
| 参与者 | 主参与者：游戏玩家  目的：在游戏开始前或结束后查看所有用户的前十名 |
| 描述 | 产生原因：用户需要查看所有用户的前十名  过程：用户点击排行榜  输出结果：系统返回成绩前十的用户的微信头像、排名以及成绩 |
| 优先级 | 10 |
| 触发条件 | 用户点击排行榜 |
| 前置条件 | 用户进入开始界面或处于游戏结束界面 |
| 后置条件 | 系统显示所有用户中前十名的排名、成绩以及微信头像 |
| 正常流程 | 1. 点击排行榜按钮  2. 系统显示所有用户中前十名的微信头像、成绩和排名 |
| 分支流程 | 无 |
| 异常流程 | 2.1 网络链接异常，系统用弹窗提示用户，头像、成绩、排名使用默认值 |
| 相关用例 | 无 |

如表3.3所示，用户可以查看所有用户中前十的成绩，并以微信头像标示出来。

表3.4 开始游戏用例表

|  |  |
| --- | --- |
| ID | UC-03 |
| 名称 | 开始游戏 |
| 参与者 | 主参与者：游戏玩家  目的：开始游戏过程 |
| 描述 | 产生原因：用户需要进入游戏过程  过程：用户点击开始游戏按钮  输出结果：游戏开始 |
| 优先级 | 10 |
| 触发条件 | 用户点击开始游戏按钮 |
| 前置条件 | 用户已经进入初始界面 |
| 后置条件 | 游戏开始 |
| 正常流程 | 1. 用户点击开始游戏 2. 系统初始化游戏数据 3. 进入游戏界面 4. 主角播放空闲状态动画 |
| 分支流程 | 无 |
| 异常流程 | 无 |
| 相关用例 | 无 |

如表3.4所示，用户开始游戏后进入游戏界面，并且播放主角空闲状态动画。

表3.5 防守用例表

|  |  |
| --- | --- |
| ID | UC-04 |
| 名称 | 防守 |
| 参与者 | 主参与者：游戏玩家  目的：成功防守获取得分 |
| 描述 | 产生原因：成功防守触发反击才能得分  过程：用户点击任何一处  输出结果：防守结果 |
| 优先级 | 10 |
| 触发条件 | 用户点击屏幕 |
| 前置条件 | 游戏已经开始 |
| 后置条件 | 根据防守结果播放动画 |
| 正常流程 | 1. 用户点击屏幕 2. 系统显示防守图标 3. 主角播放防守动画，进入防守状态 4. 防守成功，主角播放反击动画 5. 敌人被消灭 6. 系统修改当前成绩 |
| 分支流程 | 3.1 主角处于非空闲状态时防守事件被忽略  4a.1 无效防守，没有敌人处于进攻状态  4a.2 主角恢复到空闲状态，播放空闲动画  4b.1 防守失败，用户没有在敌人刚开始攻击的时候点击屏幕  4b.2 主角播放受伤动画，按照伤害量减少生命值，游戏界面生命值改变  4b.2a1 主角生命值降至0，主角播放死亡动画，游戏结束  4b.2a2 游戏进入结束界面，隐藏暂停按钮和主角生命条  4b.3 主角恢复空闲状态，播放空闲动画  5a.1 敌人生命值没有降至0，显示敌人现有生命值，敌人后退  5a.2 敌人回到攻击位置，开始攻击 |
| 异常流程 | 无 |
| 相关用例 | UC-09 |

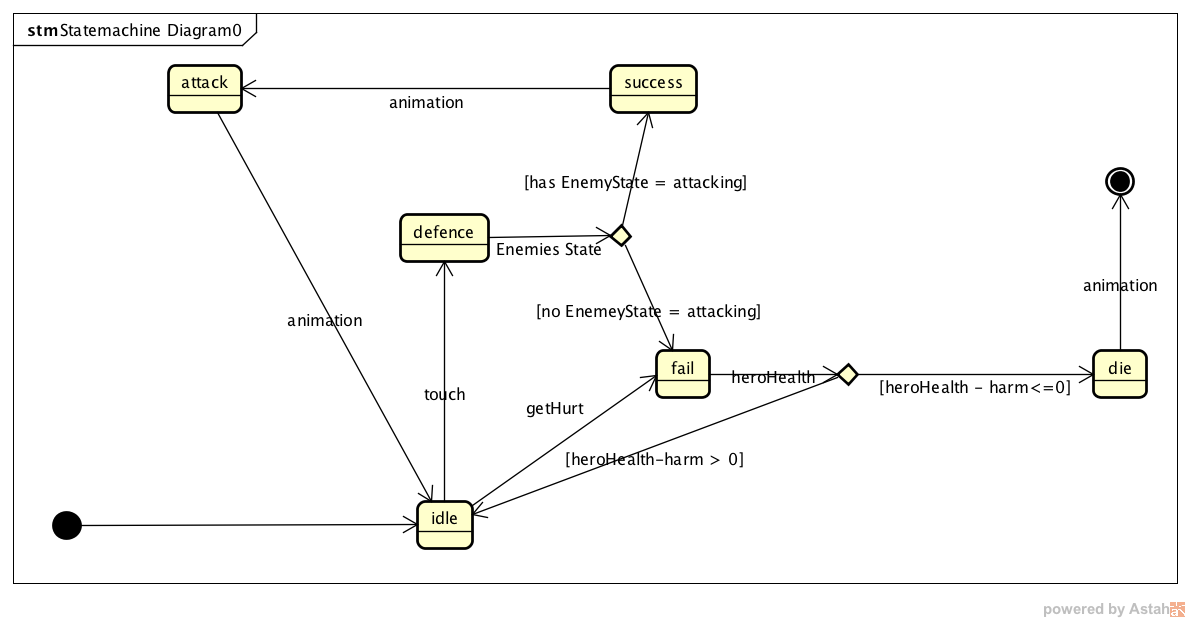


图3.3 防守状态图

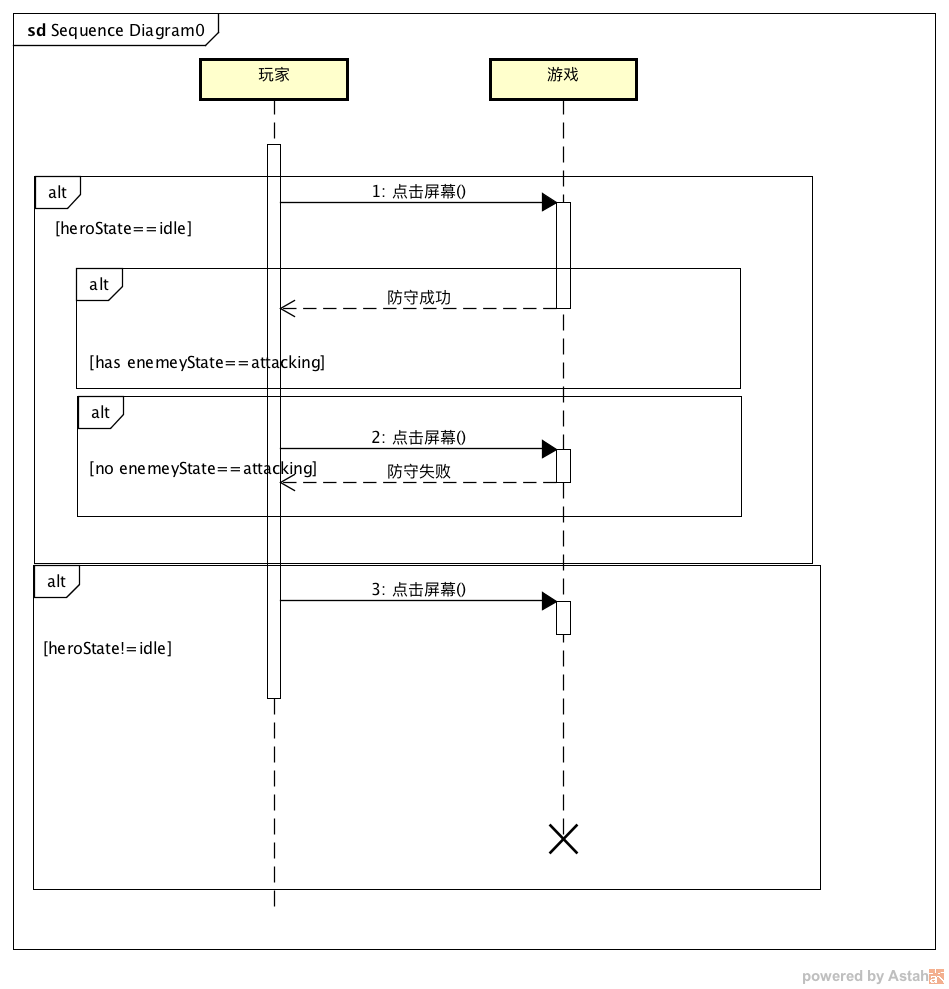


图3.4 防守顺序图

如表3.5、图3.3、图3.4所示，用户在游戏开始后，点击屏幕时，系统判断当前游戏角色是否处于空闲状态，如果是空闲状态则做出防守动作。在开始防守时判断当前屏幕中所有敌人是否有处于正在攻击的状态，如果是，则在防守结束后进入反击状态，对敌人进行攻击，敌人减少生命值并使敌人位置后退，如果敌人生命值降为0，敌人死亡。

表3.6 暂停游戏用例表

|  |  |
| --- | --- |
| ID | UC-05 |
| 名称 | 暂停游戏 |
| 参与者 | 主参与者：游戏玩家  目的：暂停游戏 |
| 描述 | 产生原因：用户在游戏过程中需要暂时中断游戏过程  过程：用户点击暂停游戏按钮  输出结果：游戏暂停 |
| 优先级 | 10 |
| 触发条件 | 用户点击暂停游戏按钮 |
| 前置条件 | 用户已经进入游戏 |
| 后置条件 | 游戏暂停 |
| 正常流程 | 1. 用户点击暂停游戏   2. 系统暂停当前画面中所有动画  3. 系统显示暂停界面，隐藏暂停按钮和主角生命条 |
| 分支流程 | 无 |
| 异常流程 | 无 |
| 相关用例 | UC-06 |

如表3.6所示，用户点击暂停后，应当隐藏游戏中的暂停按钮和主角的生命条，并且展示暂停游戏画面，暂停界面包含返回和再来一次等按钮。

表3.7 恢复游戏用例表

|  |  |
| --- | --- |
| ID | UC-06 |
| 名称 | 恢复游戏 |
| 参与者 | 主参与者：游戏玩家  目的：恢复暂停的游戏 |
| 描述 | 产生原因：用户需要从暂停游戏中恢复  过程：用户点击继续游戏按钮  输出结果：游戏恢复 |
| 优先级 | 10 |
| 触发条件 | 用户点击继续游戏按钮 |
| 前置条件 | 游戏已经暂停 |
| 后置条件 | 游戏中暂停的元素继续工作 |
| 正常流程 | 1. 用户点击继续游戏按钮 2. 游戏恢复 3. 系统显示暂停按钮和主角生命条，隐藏暂停界面 |
| 分支流程 | 无 |
| 异常流程 | 无 |
| 相关用例 | UC-05 |

如表3.7所示，用户点击继续游戏按钮，游戏恢复，并且显示主角生命条和暂停按钮，隐藏暂停界面。

表3.8 重新开始游戏用例表

|  |  |
| --- | --- |
| ID | UC-07 |
| 名称 | 重新开始游戏 |
| 参与者 | 主参与者：游戏玩家  目的：重新开始游戏 |
| 描述 | 产生原因：用户对当前成绩不满意  过程：用户点击重新开始游戏按钮  输出结果：游戏重新加载 |
| 优先级 | 10 |
| 触发条件 | 用户点击重新开始游戏按钮 |
| 前置条件 | 游戏进入暂停界面或结束界面 |
| 后置条件 | 数据重置 |
| 正常流程 | 1. 用户点击重新开始游戏按钮 2. 系统重置游戏数据 3. 游戏重新开始 |
| 分支流程 | 无 |
| 异常流程 | 无 |
| 相关用例 | UC-05 |

如表3.8所示，用户在进入暂停界面或者结束游戏后，可以选择重新开始游戏，游戏的数据应该重置。

表3.9 退出游戏用例表

|  |  |
| --- | --- |
| ID | UC-08 |
| 名称 | 退出游戏 |
| 参与者 | 主参与者：游戏玩家  目的：退出游戏 |
| 描述 | 产生原因：用户需要退出游戏  过程：用户点击退出按钮  输出结果：游戏结束并退出 |
| 优先级 | 10 |
| 触发条件 | 用户点击退出游戏按钮 |
| 前置条件 | 用户已经进入暂停界面或游戏结束 |
| 后置条件 | 退出游戏 |
| 正常流程 | 1. 用户点击退出游戏 2. 系统判断当前环境，若是web则回退页面 |
| 分支流程 | 2a.1 若是原生环境则结束游戏 |
| 异常流程 | 无 |
| 相关用例 | 无 |

如表3.9所示，用户点击退出游戏后，系统判断当前环境，进行不同的操作。

表3.10 获取用户成绩奖励用例表

|  |  |
| --- | --- |
| ID | UC-09 |
| 名称 | 获取用户成绩和奖励 |
| 参与者 | 主参与者：游戏玩家  目的：获取与成绩相对应的奖励 |
| 描述 | 产生原因：用户需要获取奖励  过程：游戏结束，系统返回对应的奖励兑换码  输出结果：奖励兑换码 |
| 优先级 | 10 |
| 触发条件 | 游戏结束 |
| 前置条件 | 用户玩过游戏并正常结束游戏 |
| 后置条件 | 可以使用兑换码 |
| 正常流程 | 1. 游戏结束 2. 系统隐藏暂停按钮和主角生命条 3. 系统显示游戏结束界面，暂停所有动画和计时器 4. 系统返回当前用户所获取的奖励兑换码 5. 系统返回当前用户的微信头像、成绩以及排名 6. 系统返回当前所有用户排行前十的微信头像、成绩与排名 |
| 分支流程 | 4a.1 系统判断当前用户成绩是否较上次有等级上提升，若没有则显示文案提示用户没有新的兑换码，显示查看兑换码按钮 |
| 异常流程 | 无 |
| 相关用例 | 无 |

如表3.10所示，游戏在结束后返回相应的奖励。

## 3.3 系统的概要设计

## 3.3.1 系统的模块划分

### 未命名:Users:sjwjames:Personal:Dionysus:file:file:subsystem.png

图3.5 系统模块划分

如图所示，《快斩格挡》游戏可以分为游戏过程控制系统、战斗系统和游戏历史系统三个子系统。战斗系统是游戏的主要模块，主要有自动刷新敌人、敌人攻击、伤害结算、主角防御、反击等功能，游戏控制系统主要包括游戏的开始结束，暂停恢复。游戏历史系统主要包括游戏成绩的增改查，排行功能。

## 3.3.2逻辑视图与开发视图

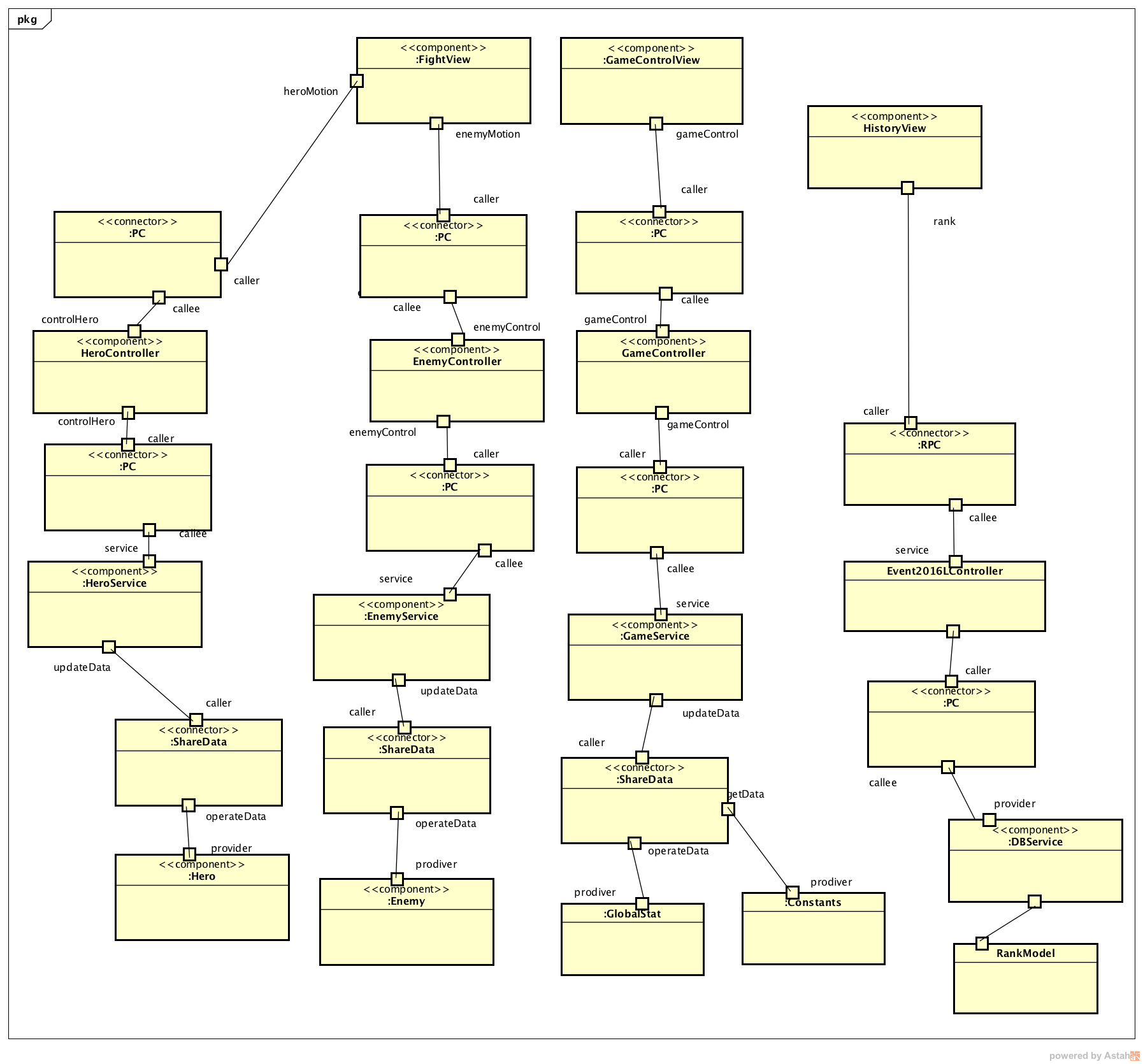


图3.6 逻辑视图

如图3.6所示，系统主要分为三个模块，分别为游戏过程控制、战斗和历史数据。三个模块都有自己的前端展示部分(FightView/GameControlView/

HistoryView)，控制跳转部分(HeroController/EnemyController/GameController

/Event2016LController)，前三个Controller位于前端，Event2016LController则在后端，系统服务层(HeroService/EnemyService/GameService)。游戏过程控制和战斗系统不设计传统数据库的读写，只有本地数据读写和浏览器数据库的读写，历史数据位于服务端，使用MySQL作为数据库来存储用户的数据，所以独有一个DBService。

如图3.7所示，游戏采用了MVC的风格来进行开发。每个模块都有自己的View，Controller，Service。

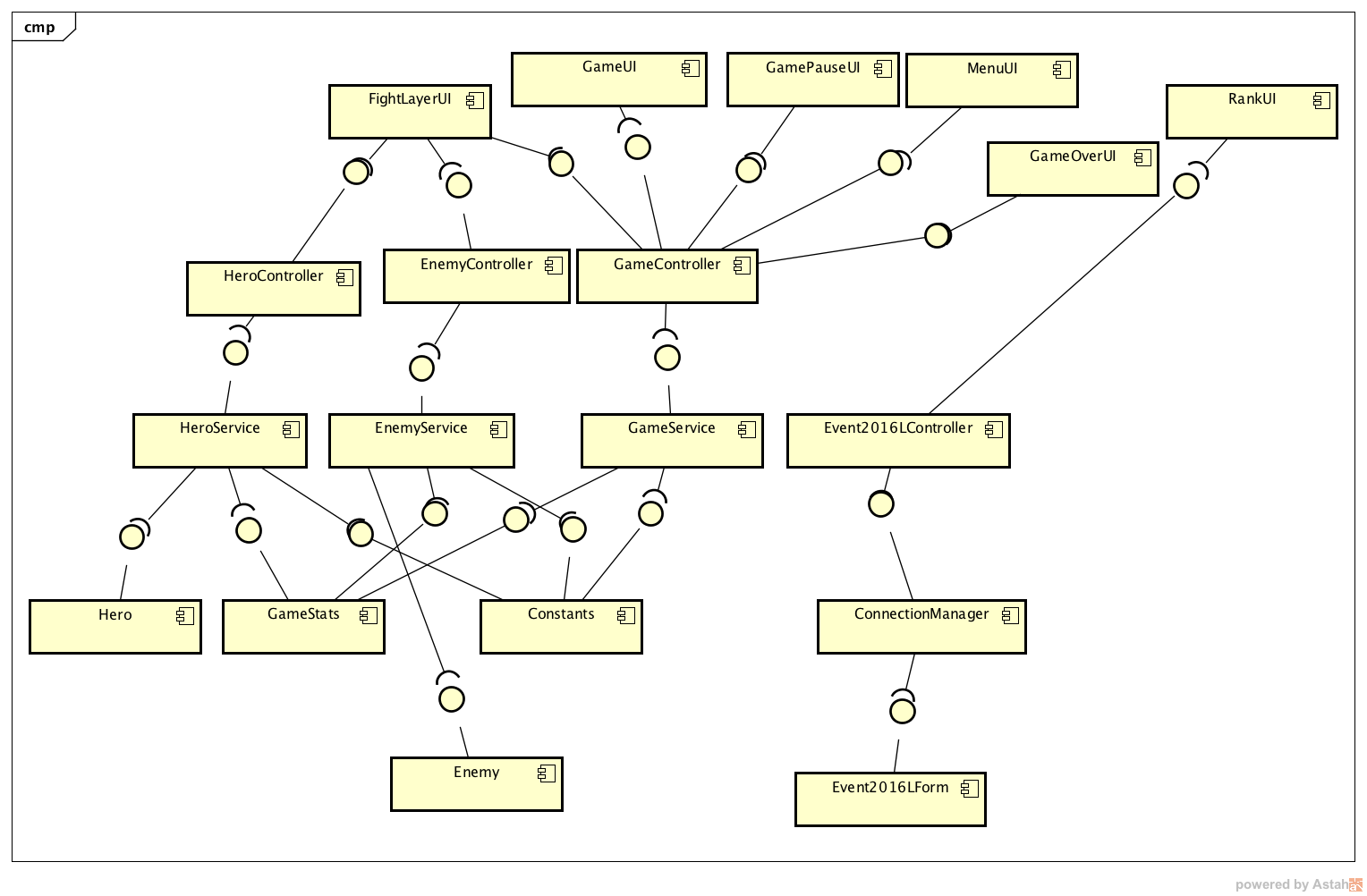


图3.7 开发视图

## 逻辑视图与开发视图的映射

表3.10 逻辑视图与开发视图映射图

|  |  |
| --- | --- |
| 部件/连接件 | 实现模块 |
| FightView | FightLayerUI |
| GameControlView | GameUI/GamePauseUI/MenuUI/GameOverUI |
| HistoryView | RankUI |
| HeroController | HeroController |
| EnemyController | EnemyController |
| GameController | GameController |
| Event2016LController | Event2016LController |
| HeroService | HeroService |
| GameService | GameService |
| EnemyService | EnemyService |
| DBService | ConnectionManager |
| Hero | Hero |
| Enemy | Enemy |
| GameStats | GameStats |
| Constants | Constants |
| RankModel | Event2016LForm |

如表3.10所示，表中逻辑视图的部件在开发视图中有一个或多个对应的模块。

## 本章小结

本章第一小节对游戏的整体和基本功能进行了概述，第二节根据功能进行了简要的需求分析，然后罗列出系统用例，对比较复杂的游戏主体进行了多个角度的需求分析，从中提取出功能需求。第三节对游戏进行了概要设计，简单地阐述了所有模块以及功能，展示了逻辑视图和开发视图，并对两者的映射关系进行了叙述。

# 第四章 快斩格挡的详细设计与实现

## 4.1 模块概述

游戏主要包括游戏控制模块、战斗模块和历史数据模块。游戏控制模块主要有控制游戏开始、暂停、恢复、暂停、重新开始等功能；战斗模块主要包括生成敌人，敌人行走、停止、攻击和死亡，主角的防守、反击、受伤和死亡，成绩统计，获取奖励码等功能；历史数据模块主要包括查看玩家成绩排名以及查看排行榜前十等功能。

## 4.2游戏控制模块和战斗模块的详细设计

## 4.2.1 游戏控制模块的详细设计

## 4.2.1.1 承担的需求

如前所述，游戏控制模块主要是负责为玩家提供进行控制游戏进程的功能。用户在打开游戏后点击进入游戏按钮将会跳转到游戏界面，并且初始化游戏。在游戏进行过程中，用户可以随时点击界面上的暂停游戏按钮，暂停游戏，并且显示相应的暂停界面，动画恢复播放，给予用户继续、重新开始、退出游戏等入口，并隐藏游戏界面中的暂停按钮和主角的生命条。用户选择继续后，暂停界面消失，恢复显示生命条和暂停按钮，游戏动画恢复播放。在用户选择重新开始后，重新加载游戏界面，重置游戏数据。用户选择退出游戏后，游戏判断当前运行环境是web环境还是原生环境，当处于web环境下时，用户跳转至之前的页面，否则退出应用。

## 游戏控制模块类图

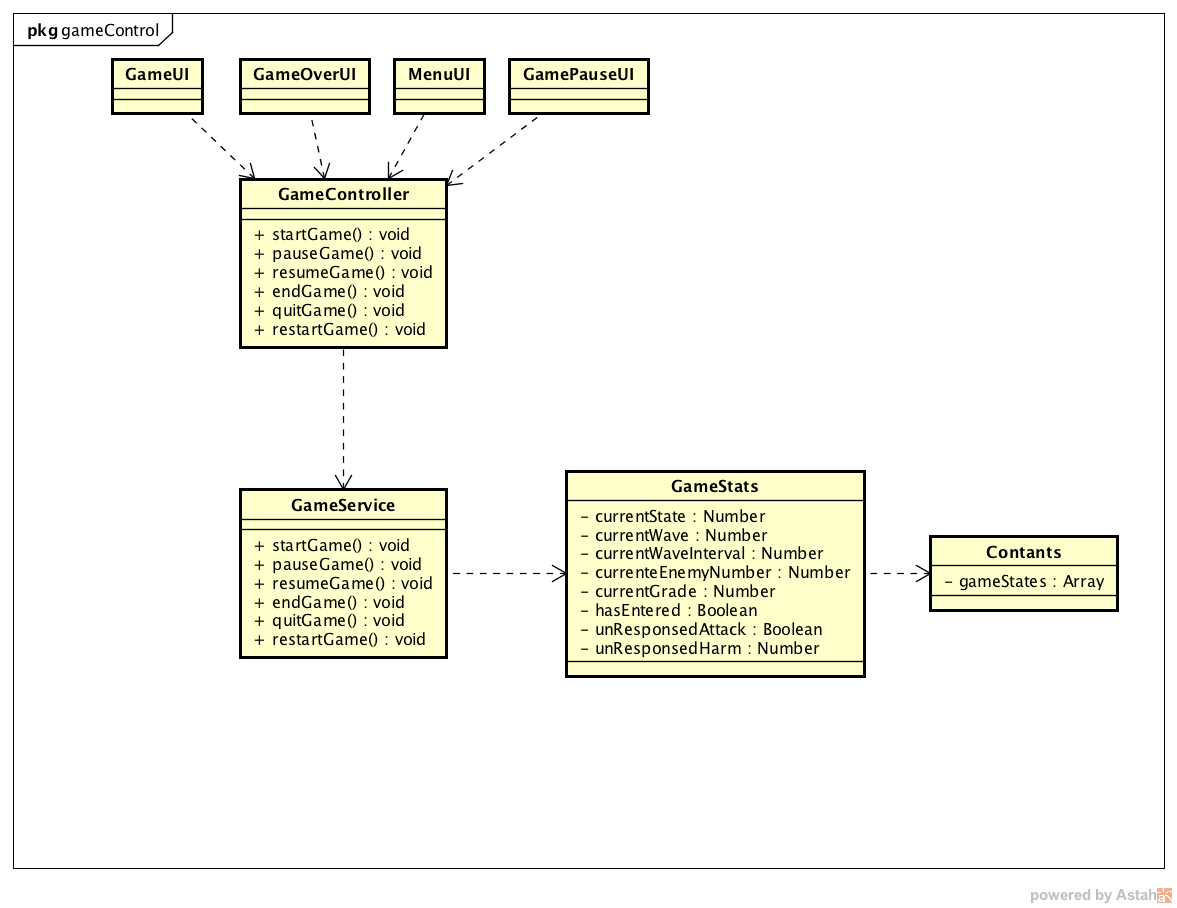


图4.1 游戏过程控制类图

如图4.1所示，游戏过程控制模块一共涉及5个类，其中Constants和GameStats为共享的数据类，在此只叙述模块中涉及到的属性，省略了get/set方法。

## 游戏控制模块的相关类描述

模块根据MVC的设计模式进行划分，UI类依赖于GameController类，GameController主要负责对用户在UI上的操作映射到GameService类中，GameService类根据不同输入对数据进行操作。此处主要涉及的是hasEntered和currentState两个属性。以下对游戏控制模块的主要类进行描述：

表4.1 GameController接口列表

|  |  |
| --- | --- |
| 接口名1 | startGame:function() |
| 接口描述 | 负责游戏开始的数据初始化和界面跳转 |
| 前置条件 | 用户点击开始游戏按钮 |
| 后置条件 | 调用GameService来初始化数据并加载游戏页面 |
| 接口名2 | pauseGame:function() |
| 接口描述 | 负责暂停游戏 |
| 前置条件 | 用户点击暂停按钮 |
| 后置条件 | 调用GameService来暂停游戏。 |
| 接口名3 | resumeGame:function() |
| 接口描述 | 负责恢复游戏 |
| 前置条件 | 游戏已经暂停，用户点击继续按钮 |
| 后置条件 | 调用GameService来恢复游戏 |
| 接口名4 | endGame:function() |
| 接口描述 | 负责结束游戏 |
| 前置条件 | 游戏正常结束，即主角生命值降至0 |
| 后置条件 | 调用GameService来结束游戏 |
| 接口名5 | quitGame:function() |
| 接口描述 | 负责退出游戏 |
| 前置条件 | 用户点击退出按钮 |
| 后置条件 | 调用GameService来退出游戏 |
| 接口名6 | restartGame:function() |
| 接口描述 | 负责重新开始游戏 |
| 前置条件 | 用户点击重新开始游戏按钮 |
| 后置条件 | 调用GameService来重新开始游戏 |

如表4.1所示，表中罗列了GameController提供的6个接口以及各个接口的描述，这些接口覆盖了对游戏进程进行控制的相关功能。

表4.2 GameService接口列表

|  |  |
| --- | --- |
| 接口名1 | startGame:function() |
| 接口描述 | 负责调用开始游戏服务 |
| 前置条件 | Controller请求开始游戏服务 |
| 后置条件 | 跳转到游戏场景 |
| 接口名2 | pauseGame:function() |
| 接口描述 | 负责调用暂停游戏服务 |
| 前置条件 | Controller请求暂停游戏服务 |
| 后置条件 | 暂停游戏，显示暂停游戏画面并隐藏生命条和暂停按钮 |
| 接口名3 | resumeGame:function() |
| 接口描述 | 负责调用恢复游戏服务 |
| 前置条件 | Controller请求恢复游戏 |
| 后置条件 | 游戏恢复，隐藏暂停游戏画面并显示生命条和暂停按钮 |
| 接口名4 | endGame:function() |
| 接口描述 | 负责调用结束游戏服务 |
| 前置条件 | Controller请求结束游戏 |
| 后置条件 | 显示游戏结束界面，显示用户的成绩与排名以及奖励。 |
| 接口名5 | quitGame:function() |
| 接口描述 | 负责调用退出游戏服务 |
| 前置条件 | Controller请求退出游戏 |
| 后置条件 | 游戏根据当前环境退出游戏或者跳转到上一个页面 |
| 接口名6 | restartGame:function() |
| 接口描述 | 负责调用重新开始游戏服务 |
| 前置条件 | Controller请求重新开始游戏 |
| 后置条件 | 游戏重新开始，数据重置 |

如表4.2，表中罗列了GameService提供的6个接口以及各个接口的描述，这些接口覆盖了对游戏进程进行控制的相关功能。

## 游戏控制模块的顺序图

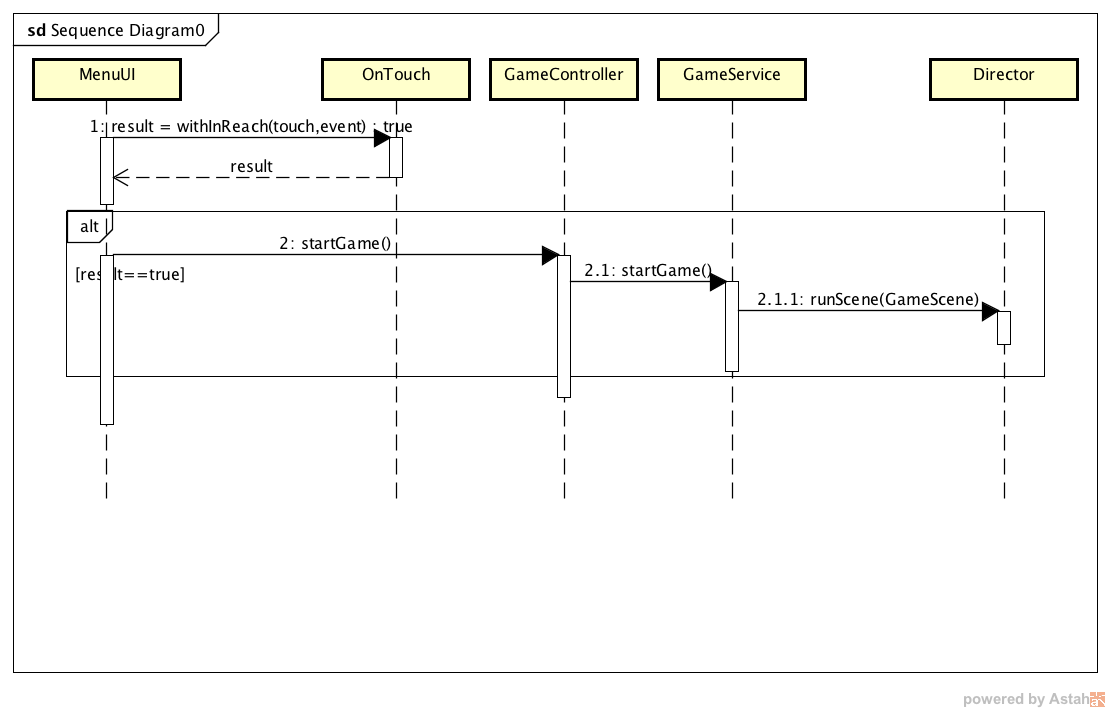


图4.2 开始游戏顺序图

如图4.2所示，当点击界面上单纯的精灵图片引发的触摸事件时由于框架自身的缺陷，需要用自己编写方法来判断触摸点是不是落在目标之内，否则点击整个屏幕都可以触发触摸事件。这个用例涉及到了切换场景的操作，故而拿出来分析一下，其实内部比较简单，在触摸点落在目标范围内时，调用GameController，GameController调用GameService的startGame方法，startGame内部调用框架的Director类中的runScene来切换场景至GameScene。重新开始游戏流程与之相似，故不赘述。

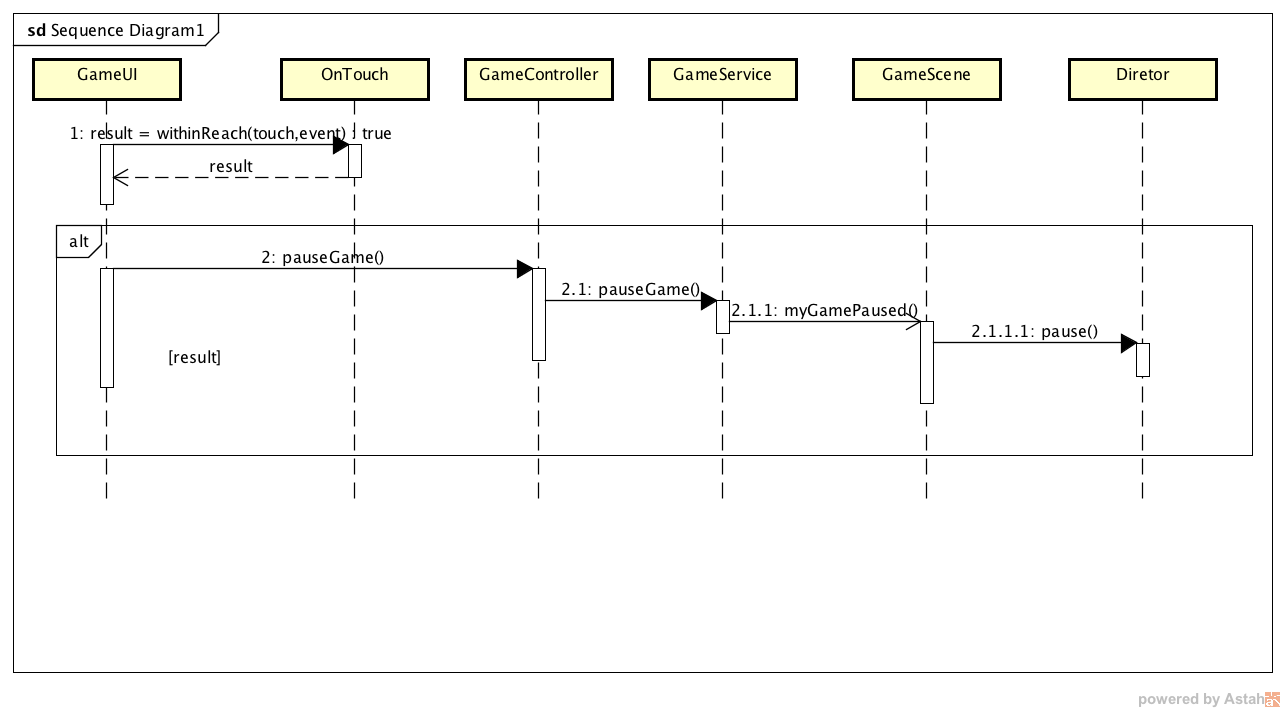


图4.3 暂停游戏顺序图

如图4.3所示，暂停游戏时，GameUI上触发触摸事件，并用OnTouch判断是否落在了按钮内部，如果是，则调用GameController的pauseGame方法，pauseGame方法会调用GameService来抛出myGamePause这样一个自定义事件，GameScene中注册了这个事件的监听之后，会调用事件的处理方法，在事件的处理方法中修改GameScene的UI，并调用框架中Director的pause方法来暂停游戏。恢复游戏流程与之相似，故不赘述。

## 战斗模块的详细设计

## 4.2.2.1 承担的需求

战斗模块承担了游戏的主要需求，亦即游戏的主体内容。战斗模块需要完成的功能有：循环播放主角空闲时动画，根据现有敌人数目自动生成敌人，敌人行走，敌人停止、攻击并根据不同防御波数调整敌人的攻击力、血量以及攻击频率，响应玩家点击屏幕事件，判断防御成功与否，显示主角防御效果，播放主角反击动画，播放主角受伤动画以及生命条变化，播放敌人受伤后退效果，敌人死亡效果，成绩结算，初次开始游戏简单教学。

## 战斗模块的类图

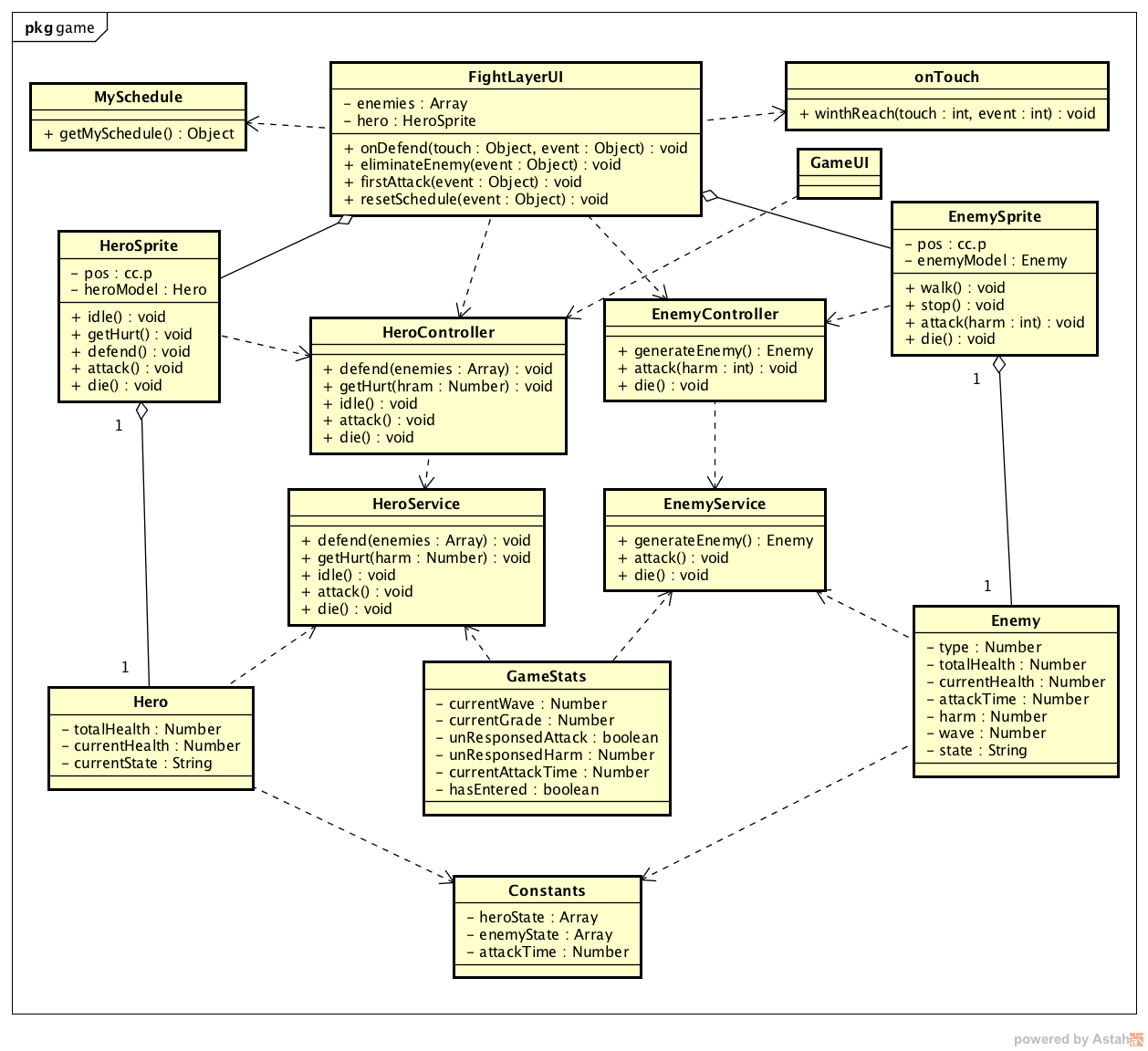


图4.4 战斗模块类图

如图4.4所示，战斗模块一共涉及11个类，其中GameStats/Constants是共享数据类，Enemy和Hero是模型数据类，接口基本只有set/get方法，故不赘述。

## 战斗模块的相关类描述

模块根据MVC的设计模式进行划分，FightLayerUI类是主战斗场景中的主要展示类，依赖于HeroController类和EnemyController类，HeroController主要负责对用户在UI上的操作以及游戏中主角的一些动作调用映射到HeroService类中，EnemyController类主要负责生成敌人并对敌人的一些动作映射到EnemyService类中。HeroService和EnemyService是主要的逻辑处理类，负责修改Hero和Enemy的属性并广播属性修改事件，HeroSprite/EnemySprite/GameUI/

FightLayerUI根据自身注册的监听来修改，HeroSprite和EnemySprite继承自框架的Sprite类，是主角和敌人的展现类，主角和敌人的动作等主要通过这两个类展示。GameUI在这个模块中主要是涉及到主角的生命值的展示和变化。

表4.3 FightLayerUI接口列表

|  |  |
| --- | --- |
| 接口名1 | onDefend:function(touch,event) |
| 接口描述 | 负责处理用户点击屏幕事件 |
| 前置条件 | 用户点击屏幕 |
| 后置条件 | 调用HeroController来处理防守事件 |
| 接口名2 | eliminateEnemy:function(event) |
| 接口描述 | 负责消除屏幕上所有的敌人 |
| 前置条件 | 用户成功防守，并进行反击 |
| 后置条件 | 调用EnemyController来清除所有敌人 |
| 接口名3 | firstAttack:function(event) |
| 接口描述 | 负责响应第一次进入游戏时用户点击屏幕事件 |
| 前置条件 | 用户第一次进入游戏，点击屏幕 |
| 后置条件 | 显示简易教程 |
| 接口名4 | resetSchedule:function(event) |
| 接口描述 | 负责响应重置定时器事件 |
| 前置条件 | 达到特定波数，加速生成敌人 |
| 后置条件 | 修改生产敌人的定时器间隔 |

如表4.3所示，FightLayerUI类的主要接口有四个，都是事件的处理函数。onDefend处理用户的点击事件，eliminateEnemy负责处理清楚敌人事件，firstAttack负责在用户第一次进行游戏的时候显示教程，resetSchedule负责重置定时器，加速生成敌人。

表4.4 OnTouch类接口列表

|  |  |
| --- | --- |
| 接口名1 | withinReach:function(touch,event) |
| 接口描述 | 负责判断用户点击是否在目标范围内 |
| 前置条件 | 用户点击屏幕 |
| 后置条件 | 返回判断结果 |

如表4.4所示，OnTouch类是一个工具类，主要是判断用户的点击事件是否落在目标范围内，由于Cocos2d-js的自身缺陷，需要这个类来辅助判断触摸事件。

表4.5 MySchedule类接口列表

|  |  |
| --- | --- |
| 接口名1 | getMySchedule:function() |
| 接口描述 | 负责生成一个有自我矫正能力的定时器 |
| 前置条件 | 需要定时器来工作 |
| 后置条件 | 返回一个有自我矫正能力的定时器 |

如表4.5所示，MySchedule类的getMySchedule返回一个定时器。由于schedule的实现基于帧频，所以当帧频下降的时候，定时器的误差会变大，schedule只能用于动画控制，不能用于逻辑处理或者网络同步的计算上。cocos2d-js自带的定时器最终都会越走越慢，因为schedule受 帧频驱动，cocos2d-js在每帧的计算中都遍历全部定时器，达到触发时间就触发定时器，但框架忽略了定时器周期和帧周期的差距问题，多次循环后误差会越来越大。

表4.5 HeroController接口列表

|  |  |
| --- | --- |
| 接口名1 | defend:function(enemies) |
| 接口描述 | 负责调用主角防守服务 |
| 前置条件 | UI请求处理用户点击屏幕事件 |
| 后置条件 | 调用HeroService类做具体的处理 |
| 接口名2 | getHurt:function() |
| 接口描述 | 负责调用主角受伤服务 |
| 前置条件 | 主角受到来自敌人的伤害 |
| 后置条件 | 调用HeroService类做逻辑处理 |
| 接口名3 | idle:function() |
| 接口描述 | 负责调用主角恢复空闲状态服务 |
| 前置条件 | 主角从其他状态中恢复 |
| 后置条件 | 调用HeroService进行处理 |
| 接口名4 | attack:function() |
| 接口描述 | 负责调用主角反击服务 |
| 前置条件 | 用户防守成功 |
| 后置条件 | 调用HeroService进行逻辑处理 |
| 接口名5 | die:function() |
| 接口描述 | 负责调用主角死亡服务 |
| 前置条件 | 主角生命值降到0 |
| 后置条件 | 调用HeroService服务 |

如表4.5所示，HeroController类主要对用户的操作或者是其他事件响应发送给HeroService类进行处理。

表4.6 HeroService接口列表

|  |  |
| --- | --- |
| 接口名1 | defend:function(enemies) |
| 接口描述 | 负责处理用户防守 |
| 前置条件 | HeroController请求处理用户防守事件 |
| 后置条件 | 触发主角防守动画 |
| 接口名2 | getHurt:function() |
| 接口描述 | 负责处理主角受伤的逻辑 |
| 前置条件 | HeroController请求处理主角受伤事件 |
| 后置条件 | 触发主角受伤动画 |
| 接口名3 | idle:function() |
| 接口描述 | 负责恢复主角空闲状态 |
| 前置条件 | HeroController请求处理恢复空闲状态 |
| 后置条件 | 根据当前状态触发恢复空闲或者是反击事件 |
| 接口名4 | attack:function() |
| 接口描述 | 负责处理主角反击事件的逻辑 |
| 前置条件 | HeroController请求处理反击事件的逻辑 |
| 后置条件 | 触发主角反击动画 |
| 接口名5 | die:function() |
| 接口描述 | 负责处理主角死亡事件 |
| 前置条件 | HeroController请求处理主角死亡的逻辑 |
| 后置条件 | 触发主角死亡动画 |

如表4.7，HeroService的5个接口都处理来自HeroController的调用。

表4.6 HeroSprite接口列表

|  |  |
| --- | --- |
| 接口名1 | defend:function() |
| 接口描述 | 负责播放防守动画 |
| 前置条件 | HeroService触发播放防守事件 |
| 后置条件 | 主角做出防守动作后UI调用HeroController的idle方法 |
| 接口名2 | getHurt:function() |
| 接口描述 | 负责播放主角受伤动画 |
| 前置条件 | HeroService触发播放受伤事件 |
| 后置条件 | 主角做出受伤动作后UI调用HeroController的idle方法 |
| 接口名3 | idle:function() |
| 接口描述 | 负责播放主角空闲状态时动画 |
| 前置条件 | HeroService触发空闲事件 |
| 后置条件 | 播放空闲状态动画 |
| 接口名4 | attack:function() |
| 接口描述 | 负责播放主角反击动画 |
| 前置条件 | HeroService触发反击事件 |
| 后置条件 | 触发清除敌人事件 |
| 接口名5 | die:function() |
| 接口描述 | 负责播放主角死亡动画 |
| 前置条件 | HeroService触发主角死亡事件 |
| 后置条件 | 游戏结束，调用GameController的endGame方法 |

如表4.6所示，HeroSprite的5个接口负责响应HeroService触发的事件，播放响应的动画并调用其他的Controller类来进行后续操作。

表4.7 EnemyController接口列表

|  |  |
| --- | --- |
| 接口名1 | generateEnemy:function() |
| 接口描述 | 负责调用EnemyService来处理生成敌人的逻辑 |
| 前置条件 | UI请求生成敌人以加入场景 |
| 后置条件 | 请求EnemyService处理生成敌人的逻辑 |
| 接口名2 | attack:function() |
| 接口描述 | 负责调用EnemyService来处理敌人攻击事件 |
| 前置条件 | 敌人移动到固定位置后 |
| 后置条件 | 请求EnemyService处理攻击的逻辑 |
| 接口名3 | die:function() |
| 接口描述 | 负责调用EnemyService处理敌人死亡的逻辑 |
| 前置条件 | 敌人生命值降低至0 |
| 后置条件 | 调用EnemyService处理敌人死亡事件 |

如表4.7所示，EnemyController主要着力于将对于敌人的操作传递到EnemyService来处理。

表4.8 EnemyService接口列表

|  |  |
| --- | --- |
| 接口名1 | generateEnemy:function() |
| 接口描述 | 负责生成敌人 |
| 前置条件 | EnemyController请求生成敌人 |
| 后置条件 | 返回一个敌人实例并添加到场景中 |
| 接口名2 | attack:function(harm) |
| 接口描述 | 负责处理敌人攻击的逻辑，根据主角状态来判断是否对主角造成伤害 |
| 前置条件 | EnemyController请求处理敌人攻击逻辑 |
| 后置条件 | 调用HeroController的getHurt()或者忽略本次攻击 |
| 接口名3 | die:function() |
| 接口描述 | 负责处理敌人死亡的逻辑 |
| 前置条件 | EnemyController请求处理敌人死亡事件 |
| 后置条件 | 触发敌人死亡的动画 |

如表4.8所示，EnemyService的三个接口都处理来自EnemyController的处理请求，一些接口触发UI事件。

表4.9 EnemySprite接口列表

|  |  |
| --- | --- |
| 接口名1 | walk:function() |
| 接口描述 | 负责使敌人运动到指定位置 |
| 前置条件 | 敌人已经被添加到场景中 |
| 后置条件 | 敌人进入停止准备攻击状态 |
| 接口名2 | stop:function() |
| 接口描述 | 负责在敌人运动到指定位置后暂停一段时间，随后开始攻击 |
| 前置条件 | 敌人运动到指定位置 |
| 后置条件 | 敌人开始攻击 |
| 接口名3 | attack:function() |
| 接口描述 | 负责播放敌人攻击动画，并调用EnemyController处理攻击带来的后续变化 |
| 前置条件 | 敌人度过准备攻击阶段 |
| 后置条件 | 调用EnemyController的attack方法 |
| 接口名4 | die:function() |
| 接口描述 | 负责展示敌人死亡的变化 |
| 前置条件 | 监听到消灭敌人的事件 |
| 后置条件 | 触发敌人死亡动画 |

如表4.9所示，EnemySprite的四个接口处理了单个敌人的生命周期中的各个活动和播放了各个时期的动画。

## 4.2.2.4 战斗模块的顺序图

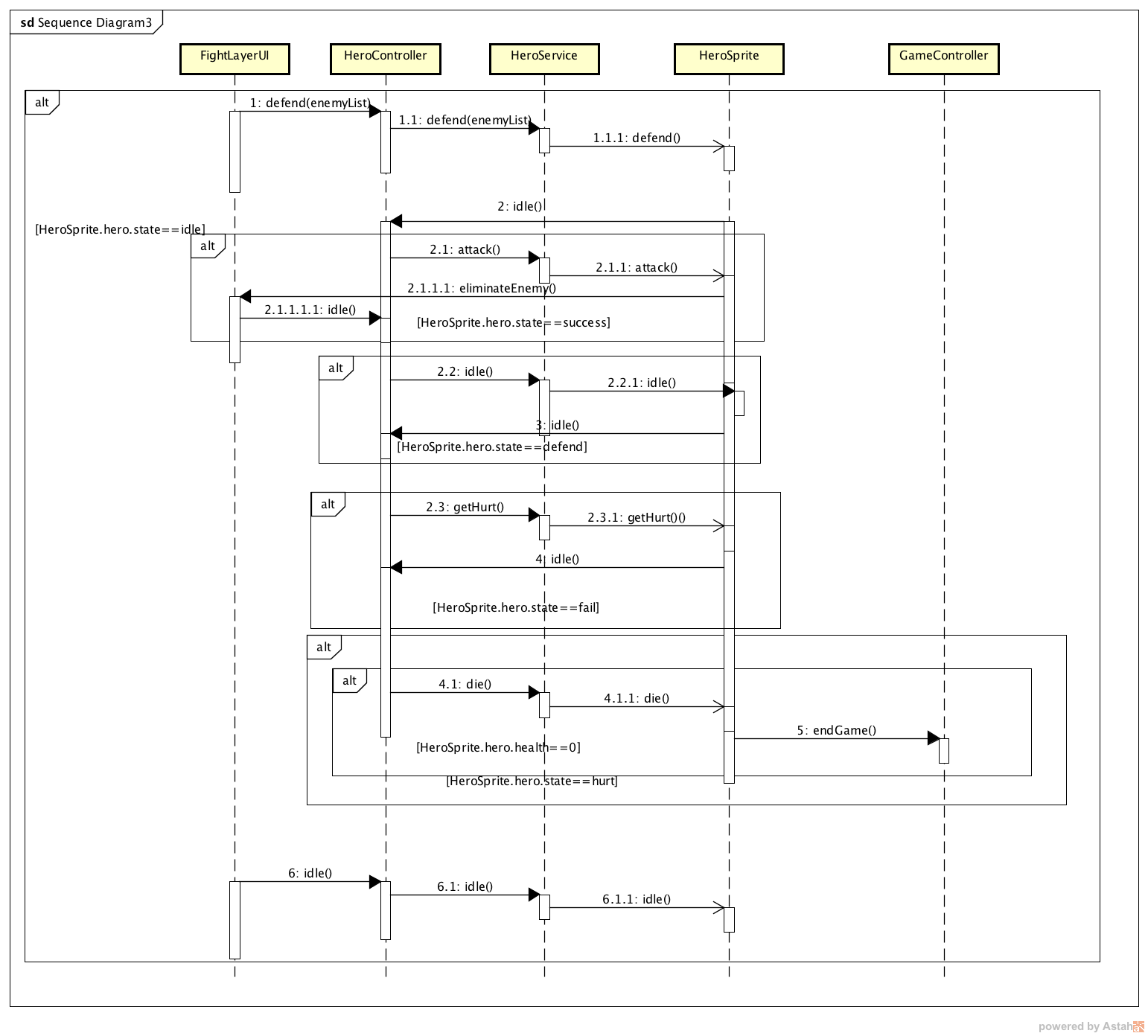


图4.5 英雄顺序图

如图4.5所示，用户点击屏幕调用HeroController的defend方法，判断当前英雄的状态如果是idle则调用HeroService的defend方法，defend触发defend事件，注册监听的HeroSprite响应事件，修改当前状态，并播放防守动画，防守动画结束后调用HeroController的idle方法，idle方法当前英雄状态来调用不同的方法。当防守成功时，FightLayerUI响应eliminateEnemy事件，显示反击动画，并清除场景中的所有敌人，调用所有敌人的die方法，此处未画出。

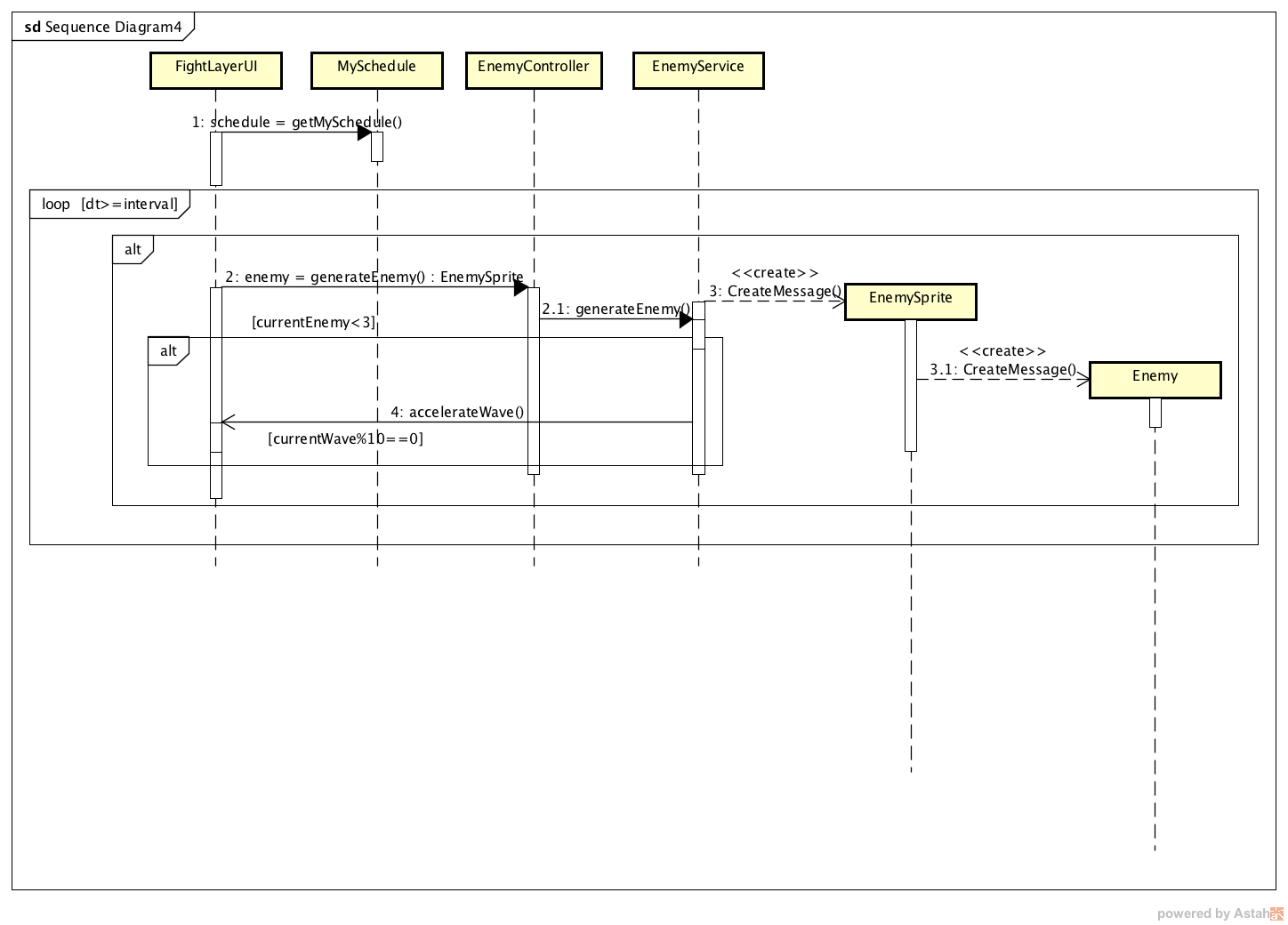


图4.6 生成敌人顺序图

如图4.6所示，FightLayerUI首先生成能够自我修正的定时器，再使用定时器定时调用EnemyController的generateEnemy方法，通过EnemyService来生成EnemySprite，当达到一定的波数后，EnemyService触发accelerateWave事件，FightLayerUI重置生成速度，亦即重置定时器的时间间隔。

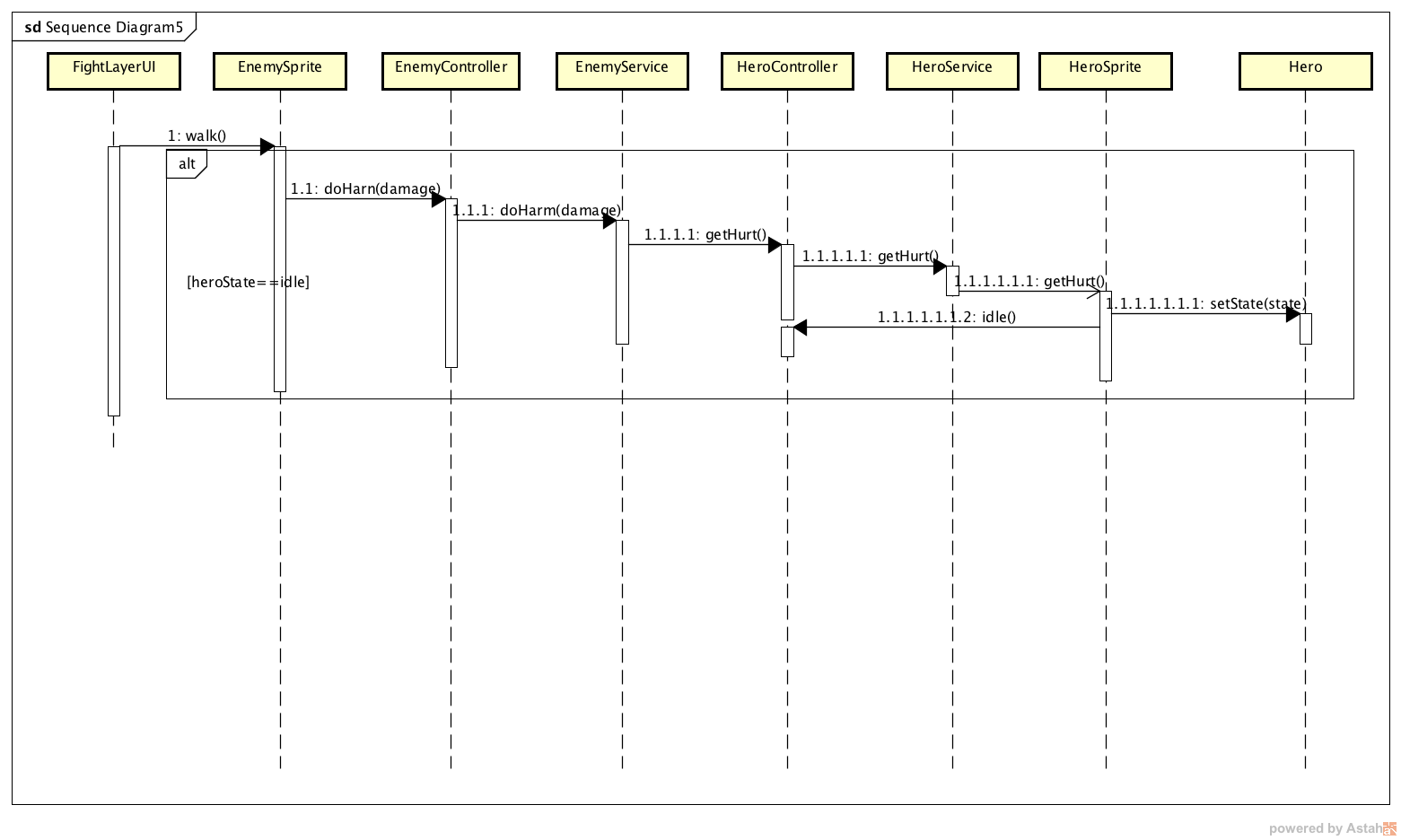


图4.7 敌人攻击顺序图

如图4.7所示，EnemySprite在被创建加入场景后，调用walk方法运动到指定位置后经过一段停顿时间后开始攻击，设置动画播放每帧更新时判断当前英雄的状态是否为idle，如果是的话则造成伤害。

## 4.3 游戏控制模块和战斗模块的实现

本节将会对游戏控制模块和战斗模块的具体实现进行详细描述，着重于介绍战斗模块的阐述。

## 4.3.1 游戏暂停、恢复

游戏的暂停恢复主要在于交替显示界面上的层，逻辑比较简单

|  |
| --- |
| onEnter:function(){  this.\_super();  cc.director.resume(); cc.eventManager.addCustomListener("myGameOver",this.onGameOver.bind(this)); cc.eventManager.addCustomListener("myGamePaused",this.onGamePaused.bind(this)); cc.eventManager.addCustomListener("myGameResumed",this.onGameResumed.bind(this)); }, onGamePaused: function (event) {  this.removeChild(this.gameUI,true);  this.pauseUI=new GamePauseUI();  this.addChild(this.pauseUI,3);  cc.eventManager.removeListener(this.fightLayer.listener);  if (this.fightLayer.getChildByTag(2)){  this.fightLayer.removeChildByTag(2);  }  if (this.fightLayer.getChildByTag(3)){  this.fightLayer.removeChildByTag(3);  }  cc.director.pause();  },  …… onExit: function () {  this.\_super();  cc.eventManager.removeCustomListeners("myGameOver");  cc.eventManager.removeCustomListeners("myGamePaused");  cc.eventManager.removeCustomListeners("myGameResumed"); } |

图4.8 GameScene类代码

如图4.8，Cocos2d的游戏元素分为场景、层次、节点等，通过将节点加入层次，层次层叠到场景，不同场景切换构成整个游戏的框架，以上的代码主要是介绍了在场景上添加层次和删除层次来展现不同的游戏过程。

## 4.3.2 生成敌人

|  |
| --- |
| generateEnemy= function (enemyList) {  if(GameStats.hasEntered){  if(GameStats.currentEnemyNumber<Constants.maxEnemy){  var enemy=EnemyService.generateEnemy();  return enemy;  }  }else{  if (enemyList.length==0){  var enemy= EnemyService.generateEnemy();  return enemy;  }else{  return null;  }  } } |

图4.9 EnemyController类代码

|  |
| --- |
| generateEnemy= function () {  if(cc.pool.hasObject(Enemy)){  console.log("from pool");  return cc.pool.\_getFromPool(Enemy);  }else{  var wave=GameStats.currentWave;  var type=wave%Constants.enemyTypes;  var imgURL="res/enemy/enemy"+(type)+"/g1.png";  GameStats.currentWave+=1;  if(GameStats.currentWave%10==0){  GameStats.currentWaveInterval=Math.pow(GameStats.currentWaveInterval,0.9);  GameStats.currentAttackTime=Math.pow(GameStats.currentAttackTime,Constants.enemyAttackAccRate);  cc.eventManager.dispatchCustomEvent("accelerateWave");  }  return new EnemySprite (imgURL,wave);  }; |

图4.9 GameService类代码

|  |
| --- |
| ctor: function () {  ……  this.generateEnemy();  this.mySchedule.mySchedule(this.generateEnemy,GameStats.currentWaveInterval,this);  },  onEnter: function () {  this.\_super(); cc.eventManager.addCustomListener("accelerateWave",this.\_resetSchedule.bind(this));  },  \_resetSchedule: function () {  this.unschedule(this.generateEnemy);  this.mySchedule.mySchedule(this.generateEnemy,GameStats.currentWaveInterval,this);  }, |

图4.10 FightLayerUI类代码

如图4.8、4.9、4.10所示，这三段是生成敌人的关键代码。在FightLayerUI类中首先在构造函数中生成了一个定时器，每GameStats.currentWaveInterval时间会调用generateEnemy函数，并且在生命周期函数onEnter中注册了accelerateWave事件的监听和处理函数\_ resetSchedule，会根据当前的生成间隔来生成敌人，EnemyService中会根据当前波数来判断是否要触发accelerateWave事件，并新建一个EnemySprite的实例返回。

## 4.3.3 防守

|  |
| --- |
| onDefend: function (touch,event) {  if (OnTouch.withInReach(touch,event)){  if(this.hero.state==Constants.heroState.idle){  var defenceSprite=new cc.Sprite("#defence.png");  defenceSprite.setPosition(touch.getLocation());  this.addChild(defenceSprite,2,2);  var fadeInAction=cc.fadeIn(0.2);  var fadeOutAction=cc.fadeOut(0.2);  var sequence=cc.sequence(fadeInAction,fadeOutAction);  defenceSprite.runAction(sequence);  var circle=new cc.Sprite("#circle.png");  circle.setPosition(touch.getLocation());  circle.scale=0.8;  this.addChild(circle,2,3);  var circleFadeIn=cc.fadeIn(0.2);  var circleFadeOut=cc.fadeOut(0.2);  var circleScale=cc.scaleTo(0.2,2,2);  var circleSpawn=cc.spawn(circleFadeOut,circleScale);  var circleSequence=cc.sequence(circleFadeIn,circleSpawn);  circle.runAction(circleSequence);  HeroController.defend(this.enemyList);  }  }  }, |

图4.11 onDefend方法实现

onDefend是FightLayerUI中屏幕触摸事件的回调函数，判断当前英雄为空闲状态时才显示响应的动画效果，并调用HeroController.defend方法来进行逻辑处理。

|  |
| --- |
| HeroService.defend=function (enemyList) {  if (GameStats.currentHeroState==Constants.heroState.idle){  GameStats.currentHeroState=Constants.heroState.defence;  for (var i=0;i<enemyList.length;i++){  if (enemyList[i].state==Constants.enemyState.attacking){  GameStats.currentHeroState=Constants.heroState.success;  break;  }  if(enemyList[i].state==Constants.enemyState.attack){  GameStats.currentHeroState=Constants.heroState.fail;  }  }  cc.eventManager.dispatchCustomEvent("heroDefence");  }else{  console.log("not prepared to defend "+GameStats.currentHeroState);  }  }; |

图4.12 HeroService.defend方法实现图

如图4.12所示，HeroService通过HeroController被调用，HeroService通过当前场景中是否有敌人处于attacking，如果有，则表示防守成功，如果没有而且没有敌人处于attack状态，则英雄状态变为defence，这个情况也就是没有敌人到位，空防守，而如果有敌人处于attack且没有处于attacking状态下的敌人则表示玩家防守慢了，失败了。

|  |
| --- |
| defence: function (event) {  this.stopAction(this.idleAction);  var heroDefenceAnimation=new cc.Animation();  for (var i=0;i<defencePics.length;i++){  heroDefenceAnimation.addSpriteFrameWithFile(defencePics[i]);  }  heroDefenceAnimation.setDelayPerUnit(Constants.defenceTime/defencePics.length);  var defenceAction = cc.animate(heroDefenceAnimation);  var shieldVisible=cc.callFunc(function () {  var fadeIn=new cc.fadeIn(0.5);  this.shield.runAction(fadeIn);  },this);  var defenceSpawn=cc.sequence(shieldVisible,defenceAction);  var reverse=defenceAction.clone().reverse();  var shieldUnvisible=cc.callFunc(function () {  this.scheduleOnce(function () {  var fadeOut=new cc.fadeOut(0.5);  this.shield.runAction(fadeOut);  }.bind(this),1);  },this);  var revereSpawn=cc.spawn(reverse,shieldUnvisible);  var callFunc=cc.callFunc(this.defenceBackToIdle,this);  var sequence=cc.sequence(defenceSpawn,revereSpawn,callFunc);  this.runAction(sequence);  }, |

图4.13 HeroSprite.defend的具体实现

如图4.13所示，HeroSprite.defend方法主要是播放英雄防守时的动画，此时英雄会做出相应动作并且身上会有类似保护罩一样的效果，并在播放完动画后调用defenceBackToIdle方法，此方法会调用HeroController.idle进行后续操作。

|  |
| --- |
| HeroController.idle= function () {  if(GameStats.currentHeroState==Constants.heroState.fail){  HeroService.getHurt();  }  if(GameStats.currentHeroState==Constants.heroState.success){  HeroService.attack();  }  if(GameStats.currentHeroState==Constants.heroState.hurt){  HeroService.idle();  }  if(GameStats.currentHeroState==Constants.heroState.idle){  HeroService.idle();  }    }; |

图4.14 HeroController.idle方法实现

如图4.14所示，HeroController.idle方法比较简单，主要是通过当前英雄状态对防守后应该做的动作进行分流。之后的Service中各个方法一次触发不同的事件，HeroSprite注册了这些事件，并通过不同的函数来播放相应的动作。

## 4.3.4 敌人攻击

|  |
| --- |
| walk: function () {  //根据当前场景中已经有的敌人数目走到指定位置，不能重叠  },  stop: function () {  this.stopAllActions();  this.state=Constants.enemyState.stop;  this.scheduleOnce(this.attack.bind(this),Constants.stopInterval);  },  attack: function () {  this.state=Constants.enemyState.attack;  //播放攻击动画  },  unattack: function () {  //攻击后摇，即准备下次攻击的一段时间间隔  },  onAttackAnimated: function () {  if(this.animationAction){  if (this.animationAction.getCurrentFrameIndex()<=Constants.successFrame){  this.state=Constants.enemyState.attacking;  if(!GameStats.hasEntered){  cc.eventManager.dispatchCustomEvent("firstAttack");  }  }else{  this.state=Constants.enemyState.attack;  if(GameStats.currentHeroState==Constants.heroState.idle){  EnemyController.doHarm(this.harm);  }    }  }  }, |

图4.15 EnemySprite类代码

如图4.15所示，EnemySprite在被加入场景后，按顺序调用walk/stop/attack，为敌人攻击动画注册一个定时器，动画每更新一帧则判断当前帧数是否在5帧以内，如果是则将当前敌人状态置为attacking，结合前面的逻辑可以知道如果此时玩家按下屏幕则会防守成功。并且感觉GameStats.hasEntered判断如果玩家当前是第一次进入游戏则会触发firstAttack事件，FightLayerUI监听了这个事件，会暂停游戏并弹出简易的教程。如果当前超过第五帧，且主角处于空闲状态，则调用EnemyController.doHarm来对主角造成伤害，最终会调用HeroService.getHurt方法。

|  |
| --- |
| getHurt: function (event) {  this.stopAction(this.idleAction);  GameStats.unResponsedHarm=0;  GameStats.unResponsedAttack=false;  var heroHurtAnimation=new cc.Animation();  for (var i=0;i<defenceFailurePics.length;i++){  heroHurtAnimation.addSpriteFrameWithFile(defenceFailurePics[i]);  }  heroHurtAnimation.setDelayPerUnit(1/defenceFailurePics.length);  var hurtAction=cc.animate(heroHurtAnimation);  var hurtCallFunc=cc.callFunc(this.failBackToIdle,this);  if(GameStats.currentHealth==0){  var dieCallFunc=cc.callFunc(this.goDie,this);  var sequence=cc.sequence(hurtAction,dieCallFunc);  this.runAction(sequence);  }else{  var sequence=cc.sequence(hurtAction,hurtCallFunc);  this.runAction(sequence);  }  } |

图4.16 HeroSprite.getHurt具体实现

如图4.16所示，HeroSprite监听了getHurt事件，当HeroService.getHurt触发getHurt事件后，HeroSprite会播放受伤动画并在结束动画后判断当前生命值是否为0，如果是则在结束后调用goDie方法，否则调用failBackToIdle方法。

## 4.4 本章小结

本章主要对游戏的过程控制模块和战斗模块的详细设计和实现进行了介绍，首先介绍了承担的需求，然后通过类图、顺序图来对这两个模块进行总体的介绍，最后对几个重要功能的逻辑代码进行了简要的介绍和分析。

# 第五章 总结与展望

## 5.1 总结

本文简要介绍了HTML5游戏背景和开发意义，列举了开发的工具和技术，较为详细地描述和分析了需求，对项目的概要设计和详细设计方面进行了比较大篇幅地叙述，力求清楚展示设计和开发的过程。

总体来说，整个项目的基本功能也就是整个游戏的基本玩法已经实现，并且具有一定的推广能力，但是仍然存在着许多不足，比如敌人的动画由于素材不足欠缺精细，在战斗中并没有来得及使用物理引擎去做碰撞检测而是用简单的使用那个状态判断，体验不佳。

## 5.2 展望

总的来说这个项目是一个不错的游戏前导推广游戏，可以满足宣传需求。如果需要更加精细地做好它，就必须要引入更加精确的物理引擎，通过人物的碰撞检测去完善战斗系统，并且加入粒子系统等来丰富整个的动画过程。

# 参考文献

[1] 郑高强著，《Cocos2d-Js开发之旅—从HTML5到原生手机游戏》， 电子工业出版社; 第1版 (2015年3月1日)，396页

[2] Nicolas C.Zakas著，李松峰 曹力译，《Javascript高级程序设计》(第三版)，人民邮电出版社，730页

[3] JS继承体系，<http://ejohn.org/blog/simple-javascript-inheritance/>

[4] 游戏数据报告，<https://www.dataeye.com/report.html>

[5] OpenGL，https://zh.wikipedia.org/wiki/OpenGL

# 致谢

感谢郑滔老师在百忙之中给予我真诚的帮助，为我审阅论文，指点迷津，给我宝贵的建议和大力支持。

同时也感谢心动网络，为我提供了宝贵的实习机会和工作岗位。尤其感谢各位同事，在工作过程中给我的无私帮助以及各种建议，使我明白了在开发过程中及时沟通的重要性，在需求不断变化的时候如何才能保证软件的健壮和灵活依旧是我需要继续学习的所在。

最后要感谢我的家人、各位师长、三位室友沈鸿斌、沈佳楠、沈杰和各位朋友，是你们不断鞭策我进步和成长，感谢南大，让我在这个四年内对大学、人生、知识有些许领悟，以学习为终身事业，以诚朴雄伟，励学敦行之校训为人生信条。