**血小板的人体冒险**

**实现阶段文档**



G15小组

组长：孙文韬

组员：韩旭、沈路通

目录

[1. 相关信息 3](#_Toc10284375)

[2. 程序清单 3](#_Toc10284376)

[2.1. 加载屏幕部分 3](#_Toc10284377)

[2.1.1. 开始游戏 3](#_Toc10284378)

[2.1.2. 加载排行榜 3](#_Toc10284379)

[2.1.3. 退出游戏 4](#_Toc10284380)

[2.2. 游戏界面部分 4](#_Toc10284381)

[2.2.1. 角色控制及碰撞 4](#_Toc10284382)

[2.2.2. 计分 10](#_Toc10284383)

[2.2.3. 镜头跟踪 11](#_Toc10284384)

[2.2.4. 死亡 12](#_Toc10284385)

[2.2.5. 怪物移动 12](#_Toc10284386)

[2.2.6. 胜利 13](#_Toc10284387)

[3. 代码规范及代码走查 14](#_Toc10284388)

[3.1. 代码规范 14](#_Toc10284389)

[3.2. 代码走查 14](#_Toc10284390)

[4. 测试 15](#_Toc10284391)

[4.1. 单元测试 15](#_Toc10284392)

[4.1.1. 模块接口 15](#_Toc10284393)

[4.1.2. 局部数据结构 15](#_Toc10284394)

[4.1.3. 重要的执行通路 15](#_Toc10284395)

[4.1.4. 出错处理通路 15](#_Toc10284396)

[4.1.5. 边界条件 15](#_Toc10284397)

[4.2. 集成测试 15](#_Toc10284398)

[4.3. 系统测试 15](#_Toc10284399)

[4.3.1. 测试范围 16](#_Toc10284400)

[4.3.2. 测试环境与系统配置 16](#_Toc10284401)

[4.4. 测试的工具及用户反馈 16](#_Toc10284402)

[4.4.1. 测试工具 16](#_Toc10284403)

[4.4.2. 用户反馈 16](#_Toc10284404)

# 相关信息

# 程序清单

## 加载屏幕部分

### 开始游戏

cc.Class({

extends: cc.Component,

properties: {

},

// 开始游戏

onLoad: function () {

this.node.on(cc.Node.EventType.TOUCH\_START, function ()

{

cc.director.loadScene("游戏界面");

});

},

});

### 加载排行榜

cc.Class({

extends: cc.Component,

properties: {

},

// 加载排行榜地图

onLoad: function () {

this.node.on(cc.Node.EventType.TOUCH\_START, function ()

{

cc.director.loadScene("排行榜");

});

},

});

### 退出游戏

cc.Class({

extends: cc.Component,

properties: {

},

// 退出游戏

onLoad: function () {

this.node.on(cc.Node.EventType.TOUCH\_START, function ()

{

cc.game.end();

});

},

});

## 游戏界面部分

### 角色控制及碰撞

cc.Class({

extends: cc.Component,

//定义属性

properties: {

//定义角色的速度，以坐标表示，有x和y方向速度

speed: cc.v2(0, 0),

movespeed:400,

//设置按住键位一直加速所能达到的最高速度

maxSpeed: cc.v2(2000, 2000),

//设置重力，出现浮空的情况可以让角色以该加速度落地

gravity: -1000,

//设置摩擦力，使得松开键位角色缓缓停止

drag: 1000,

//设置方向的flag用于判断朝向

direction: 0,

//设置跳起向上的速度

jumpSpeed: 300,

//管理停止layout的出现与消失

stopScene:{

type:cc.Layout,

default:null,

},

stopflag:false,

count:0,

},

// use this for initialization

onLoad: function () {

//动画flag管理动画播放

this.animflag=false;

this.directionright=true;

//add keyboard input listener to call turnLeft and turnRight

//开启键盘监听，获取输入的键位

cc.systemEvent.on(cc.SystemEvent.EventType.KEY\_DOWN, this.onKeyPressed, this);

cc.systemEvent.on(cc.SystemEvent.EventType.KEY\_UP, this.onKeyReleased, this);

//碰撞体x和y方向管理

this.collisionX = 0;

this.collisionY = 0;

//设置之前位置产生反弹

this.prePosition = cc.v2();

this.preStep = cc.v2();

//设置碰撞次数

this.touchingNumber = 0;

//设置暂停界面可见性

this.stopScene.node.active=false;

},

//开启碰撞体监听

onEnable: function () {

cc.director.getCollisionManager().enabled = true;

cc.director.getCollisionManager().enabledDebugDraw = false;

},

//关闭碰撞体监听

onDisable: function () {

cc.director.getCollisionManager().enabled = false;

cc.director.getCollisionManager().enabledDebugDraw = false;

},

//监听到按下事件后执行的方法

onKeyPressed: function (event) {

var anim = this.getComponent(cc.Animation);

let keyCode = event.keyCode;

switch(keyCode) {

case cc.macro.KEY.a:

case cc.macro.KEY.left:

this.direction = -1;

if(this.animflag==false){

anim.play('角色左移');

var animState =anim.play('角色左移');

animState.speed=2.5;

this.animflag=true;

this.directionright=false;

}

break;

case cc.macro.KEY.d:

case cc.macro.KEY.right:

this.direction = 1;

if(this.animflag==false){

anim.play('角色右移');

var animState =anim.play('角色右移');

animState.speed=2.5;

this.animflag=true;

this.directionright=true;

}

break;

case cc.macro.KEY.w:

case cc.macro.KEY.up:

if (!this.jumping) {

this.jumping = true;

this.speed.y = this.jumpSpeed;

}

if(this.animflag==false){

if(this.directionright==true)

anim.play('角色右跳跃');

else if(this.directionright==false)

anim.play('角色左跳跃');

this.animflag=true;

}

break;

default:

this.count++;

if(this.count==1){

this.stopflag=true;

this.stopScene.node.active=true;

}

else{

this.stopflag=false;

this.stopScene.node.active=false;

cc.director.resume();

this.count=0;

}

break;

}

},

onKeyReleased: function (event) {

var anim = this.getComponent(cc.Animation);

let keyCode = event.keyCode;

switch(keyCode) {

case cc.macro.KEY.a:

case cc.macro.KEY.left:

case cc.macro.KEY.d:

case cc.macro.KEY.right:

anim.pause('角色右移');

anim.pause('角色左移');

if(this.animflag==true){

if(this.directionright==true)

anim.play('右停止');

else if(this.directionright==false)

anim.play('左停止');

}

this.direction = 0;

this.animflag=false;

break;

case cc.macro.KEY.w:

case cc.macro.KEY.up:

this.animflag=false;

break;

}

},

//碰撞体进入的监听事件执行的方法

onCollisionEnter: function (other, self) {

this.touchingNumber ++;

// 1st step

// 获取世界坐标

var otherAabb = other.world.aabb;

var otherPreAabb = other.world.preAabb.clone();

var selfAabb = self.world.aabb;

var selfPreAabb = self.world.preAabb.clone();

// 2nd step

// 检查x坐标

selfPreAabb.x = selfAabb.x;

otherPreAabb.x = otherAabb.x;

if (cc.Intersection.rectRect(selfPreAabb, otherPreAabb)) {

if (this.speed.x < 0 && (selfPreAabb.xMax > otherPreAabb.xMax)) {

//this.node.x = otherPreAabb.xMax - this.node.parent.x;

this.collisionX = -1;

}

else if (this.speed.x > 0 && (selfPreAabb.xMin < otherPreAabb.xMin)) {

// this.node.x = otherPreAabb.xMin - selfPreAabb.width - this.node.parent.x;

this.collisionX = 1;

}

if(other.node.group==='Platform')

this.speed.x = 0;

other.touchingX = true;

return;

}

// 3rd step

// 检查y坐标

selfPreAabb.y = selfAabb.y;

otherPreAabb.y = otherAabb.y;

if (cc.Intersection.rectRect(selfPreAabb, otherPreAabb)) {

if (this.speed.y < 0 && (selfPreAabb.yMax > otherPreAabb.yMax)) {

this.node.y = otherPreAabb.yMax - this.node.parent.y;

this.jumping = false;

this.collisionY = -1;

}

else if (this.speed.y > 0 && (selfPreAabb.yMin < otherPreAabb.yMin)) {

this.node.y = otherPreAabb.yMin - selfPreAabb.height - this.node.parent.y;

this.collisionY = 1;

}

this.speed.y = 0;

other.touchingY = true;

}

},

//判断是否离开碰撞区域，即主角跳起

onCollisionExit: function (other) {

this.touchingNumber --;

if (other.touchingX) {

this.collisionX = 0;

other.touchingX = false;

}

else if (other.touchingY) {

other.touchingY = false;

this.collisionY = 0;

this.jumping = true;

}

},

//更新主角位置

update: function (dt) {

//判断是否暂停游戏

if(this.stopflag==true){

cc.director.pause();

}

//判断是否跳起

if (this.collisionY === 0) {

this.speed.y += this.gravity \* dt;

if (Math.abs(this.speed.y) > this.maxSpeed.y) {

this.speed.y = this.speed.y > 0 ? this.maxSpeed.y : -this.maxSpeed.y;

}

}

//如果判断角色停止，则通过摩擦力进行减速

if (this.direction === 0) {

if (this.speed.x > 0) {

this.speed.x -= this.drag \* dt;

if (this.speed.x <= 0) this.speed.x = 0;

}

else if (this.speed.x < 0) {

this.speed.x += this.drag \* dt;

if (this.speed.x >= 0) this.speed.x = 0;

}

}

else {

this.speed.x += (this.direction > 0 ? 1 : -1) \* this.drag \* dt;

if (Math.abs(this.speed.x) > this.maxSpeed.x) {

this.speed.x = this.speed.x > 0 ? this.maxSpeed.x : -this.maxSpeed.x;

}

}

if (this.speed.x \* this.collisionX > 0) {

this.speed.x = 0;

}

//更新之前点位置

this.prePosition.x = this.node.x;

this.prePosition.y = this.node.y;

this.preStep.x = this.speed.x \* dt;

this.preStep.y = this.speed.y \* dt;

//更新当前角色位置

this.node.x += this.speed.x \* dt;

this.node.y += this.speed.y \* dt;

},

});

### 计分

cc.Class({

extends: cc.Component,

properties: {

//对应当前发生碰撞的目标

target: cc.Node,

//对应屏幕上的label节点

label:{

type:cc.Label,

default:null,

},

},

// LIFE-CYCLE CALLBACKS:

onCollisionEnter: function (other, self) {

this.target.destroy();

this.label.count++;

this.label.string='分数: '+ this.label.count;

},

onLoad () {

},

start () {

this.label.count=0;

},

update (dt) {

},

});

### 镜头跟踪

cc.Class({

extends: cc.Component,

editor: {

requireComponent: cc.Camera, //前置要求摄像机组件

},

extends: cc.Component,

properties: {

target: {

default: null,

type: cc.Node

}

},

start () {

//获取节点上的摄像机组件

this.camera = this.getComponent(cc.Camera);

},

update(dt) {

// target到哪里，camera就到哪里

/\*\*

\* 1、target坐标转换成世界坐标

\* 2、target的世界坐标转化为camera父节点的坐标系下

\* 3、设置camera坐标

\*/

var wpos = this.target.convertToWorldSpaceAR(cc.v2(0, 0));

var pos = this.node.parent.convertToNodeSpaceAR(wpos);

// this.node.setPosition(pos);

this.node.x = pos.x+300; // 此方法只移动x轴

this.node.y=320;

},

});

### 死亡

cc.Class({

extends: cc.Component,

properties: {

},

//监测是否碰撞

onCollisionEnter: function (other, self) {

cc.director.loadScene("死亡界面");

},

onLoad () {

},

start () {

},

update (dt) {

},

});

### 怪物移动

cc.Class({

extends: cc.Component,

properties: {

speed:250,

},

// LIFE-CYCLE CALLBACKS:

// 监测是否发生碰撞，碰撞则主角死亡

onCollisionEnter: function (other, self) {

cc.director.loadScene("死亡界面");

},

start () {

},

update (dt) {

this.node.x+=this.speed\*dt;

},

});

### 胜利

cc.Class({

extends: cc.Component,

properties: {

},

//监测是否发生碰撞

onCollisionEnter: function (other, self) {

this.scheduleOnce(function() {

// 这里的 this 指向 component

cc.director.loadScene("游戏胜利");

}, 0.5);

},

onLoad () {

},

start () {

},

update (dt) {

},

});

# 代码规范及代码走查

## 代码规范

1.Cocos中场景和精灵的命名采用中文

2.变量的定义使用该中文对应的英文或节点名

3.方法之间空一行区分，每个部分加注释，以便增加代码可读性

4.=号两边不留空格

5.用于命名的单词开头要大写

6.与Cocos本身冲突的部分优先按照Cocos

7.变量名、函数名，都只敲一遍，以后全部是拷贝+粘贴。可防止因少或多打一两个字母，而出现的bug问题。

8. 变量置于块的开始处，不要总是在第一次使用它们的地方做声明。若此变量并非只在局部被使用一次，其声明就应该放在本块的开始部分，方法内部较靠前的位置进行变量的声明，易于查看和维护。

## 代码走查

我们组内每周会进行代码走查检查各自部分的代码进度以及代码是否符合规范。

|  |  |  |
| --- | --- | --- |
| **问题** | **是** | **否** |
| **总体** | √ |  |
| 代码编制是否遵照编码规范？ | √ |  |
| 缺陷修改是否完全完成？ |  | √ |
| 所有的代码是否风格保持一致？ | √ |  |
| **注释** |  |  |
| 所有的注释是否是最新的？ | √ |  |
| 所有的注释是否清楚和正确？ | √ |  |
| 是否按照注释类型格式编写注释？ | √ |  |
| 若代码修改注释是否方便修改？ | √ |  |
| 每一功能目的是否都有注释？ | √ |  |
| **源代码质量** |  |  |
| 所有变量的命名是否依据规则？ | √ |  |
| 循环嵌套是否优化到最少？ | √ |  |
| 所有代码是否易懂？ | √ |  |
| 所有涉及要求是否都实现？ | √ |  |

# 测试

## 单元测试

以下均采用了白盒测试技术的逻辑覆盖。

### 模块接口

由于我们服务器部分还没完全做好，暂时无法测试游戏本体与服务器的接口以及数据库与服务器的接口。

### 局部数据结构

我们会根据各个局部模块的功能，单独使用该模块的功能进行测试。

### 重要的执行通路

我们从项目打开的界面开始从头到尾各个功能全部运行以便进行测试。

### 出错处理通路

当项目运行出错时我们可以使用ESC直接退出并根据出错的地方进行修改。

### 边界条件

我们测试了人物不吃道具直接死亡，吃掉全部道具后死亡，不吃道具胜利以及吃掉全部道具后胜利四种边界情况。

## 集成测试

由于服务器的问题尚未完全解决，我们只做了游戏本体各模块之间的集成，整个游戏本体运行除了加载较慢外无较大问题。

## 系统测试

由于服务器的问题尚未完全解决，我们暂时无法进行完整的系统测试。

### 测试范围

(1)所有基本页面的链接:进入游戏后，检测所有页面是否正常显示。

(2)所有页面的转移正确:进入游戏后，检测所有页面是否跳转正确。

(3)登录页面:进入登录页面，输入数据，检测对输入数据进行验证。

(4)用户进行游戏:用户开始游戏之后，检测是否能正常进行游戏操作。

(5)用户查看排行:用户完成游戏之后，检测是否能正常查看分数排行。

### 测试环境与系统配置

测试环境：单机状态下

系统配置：windows10

浏览器：Google Chrome

## 测试的工具及用户反馈

### 测试工具

我们游戏方面的代码全部使用引擎自带的编译器进行测试。

### 用户反馈

我们让统计1702的金同学在网页上打开并试玩了下游戏，他表示主要有以下问题：

1. 打开游戏时等待较久，且由于没有进度条缺少反馈感
2. 有时游戏动画会出错
3. 人物吃道具时会有较明显的碰撞感
4. 主要的功能都能实现，但地图太小缺少游戏体验