**血小板的人体冒险**

**详细设计报告（ISO）**



G15小组

组长：孙文韬

组员：韩旭、沈路通

目录

[1. 引言 5](#_Toc12183733)

[1.1. 编写目的 5](#_Toc12183734)

[1.2. 背景 5](#_Toc12183735)

[1.2.1. 名称 5](#_Toc12183736)

[1.2.2. 本项目的提出者、开发者、用户 5](#_Toc12183737)

[1.3. 定义 5](#_Toc12183738)

[1.4. 参考资料 6](#_Toc12183739)

[2. 系统的结构 6](#_Toc12183740)

[3. 服务器系统模块设计说明 8](#_Toc12183741)

[3.1. 模块描述 8](#_Toc12183742)

[3.2. 功能 8](#_Toc12183743)

[3.3. 性能 8](#_Toc12183744)

[3.4. 输入项 8](#_Toc12183745)

[3.5. 输出项 8](#_Toc12183746)

[3.6. 设计方法（算法） 8](#_Toc12183747)

[3.7. 流程逻辑 8](#_Toc12183748)

[3.8. 接口 9](#_Toc12183749)

[3.9. 存储分配 9](#_Toc12183750)

[3.10. 注释设计 9](#_Toc12183751)

[3.11. 限制条件 9](#_Toc12183752)

[3.12. 测试计划 9](#_Toc12183753)

[3.13. 尚未解决的问题 9](#_Toc12183754)

[4. 数据库模块设计说明 9](#_Toc12183755)

[4.1. 模块描述 9](#_Toc12183756)

[4.2. 功能 10](#_Toc12183757)

[4.3. 性能 10](#_Toc12183758)

[4.4. 输入项 10](#_Toc12183759)

[4.5. 输出项 10](#_Toc12183760)

[4.6. 设计方法（算法） 10](#_Toc12183761)

[4.7. 流程逻辑 14](#_Toc12183762)

[4.8. 接口 14](#_Toc12183763)

[4.9. 存储分配 14](#_Toc12183764)

[4.10. 注释设计 14](#_Toc12183765)

[4.11. 限制条件 14](#_Toc12183766)

[4.12. 测试计划 14](#_Toc12183767)

[4.13. 尚未解决的问题 14](#_Toc12183768)

[5. 角色控制模块设计说明 14](#_Toc12183769)

[5.1. 模块描述 14](#_Toc12183770)

[5.2. 功能 15](#_Toc12183771)

[5.3. 性能 15](#_Toc12183772)

[5.4. 输入项 15](#_Toc12183773)

[5.5. 输出项 15](#_Toc12183774)

[5.6. 设计方法（算法） 15](#_Toc12183775)

[5.7. 流程逻辑 21](#_Toc12183776)

[5.8. 接口 21](#_Toc12183777)

[5.9. 存储分配 21](#_Toc12183778)

[5.10. 注释设计 22](#_Toc12183779)

[5.11. 限制条件 22](#_Toc12183780)

[5.12. 测试计划 22](#_Toc12183781)

[5.13. 尚未解决的问题 22](#_Toc12183782)

[6. 场景模块设计说明 22](#_Toc12183783)

[6.1. 模块描述 22](#_Toc12183784)

[6.2. 功能 22](#_Toc12183785)

[6.3. 性能 22](#_Toc12183786)

[6.4. 输入项 22](#_Toc12183787)

[6.5. 输出项 23](#_Toc12183788)

[6.6. 设计方法（算法） 23](#_Toc12183789)

[6.7. 流程逻辑 41](#_Toc12183790)

[6.8. 接口 41](#_Toc12183791)

[6.9. 存储分配 41](#_Toc12183792)

[6.10. 注释设计 41](#_Toc12183793)

[6.11. 限制条件 41](#_Toc12183794)

[6.12. 测试计划 42](#_Toc12183795)

[6.13. 尚未解决的问题 42](#_Toc12183796)

[7. 多媒体模块 42](#_Toc12183797)

[7.1. 模块描述 42](#_Toc12183798)

[7.2. 功能 42](#_Toc12183799)

[7.3. 性能 42](#_Toc12183800)

[7.4. 输入项 42](#_Toc12183801)

[7.5. 输出项 42](#_Toc12183802)

[7.6. 设计方法（算法） 42](#_Toc12183803)

[7.7. 流程逻辑 45](#_Toc12183804)

[7.8. 接口 45](#_Toc12183805)

[7.9. 存储分配 45](#_Toc12183806)

[7.10. 注释设计 45](#_Toc12183807)

[7.11. 限制条件 45](#_Toc12183808)

[7.12. 测试计划 45](#_Toc12183809)

[7.13. 尚未解决的问题 45](#_Toc12183810)

# 引言

## 编写目的

我们在进行了一定量的调查后发现包括我们自身在内的许多大学生都对游戏感兴趣，除了周末之外我们的空余时间都太过碎片化，而且课后也有一定量的学习任务，不能完全用来玩游戏也更不要说去玩一些一盘就是几十分钟的游戏了。所以我们想要设计一款游戏时间较短但又能有一定趣味性与可玩性的游戏，而目前市面上比较流行的游戏有生存类的、休闲类的、战略养成类的等等。反倒是过去较火的横版闯关游戏有些少见，因此我们想要为喜欢这类游戏但又没有较多娱乐时间的人们做一款游戏。同时前段时间《工作细胞》这部动漫大火，吸引了大量的粉丝，我们以这部动漫作为游戏背景也能吸引一部分这个动漫的粉丝。

## 背景

### 名称

血小板的冒险

### 本项目的提出者、开发者、用户

提出者及开发者：孙文韬、韩旭、沈路通

用户：忙于学习任务，没有充足时间玩大型游戏的人群，主要以大学生为主。

## 定义

a.HTML5：万维网的核心语言、标准通用标记语言下的一个应用超文本标记语言（HTML）的第五次重大修改。

b.Cocos：Cocos是由触控科技推出的游戏开发一站式解决方案，包含了从新建立项、游戏制作、到 打包上线的全套流程。开发者可以通过Cocos快速生成代码、编辑资源和动画，最终输出适合于多个平台的游戏产品

c. JavaScript：JavaScript一种直译式脚本语言，是一种动态类型、弱类型、基于原型的语言，内置支持类型。它的解释器被称为JavaScript引擎，为浏览器的一部分，广泛用于客户端的脚本语言，最早是在HTML（标准通用标记语言下的一个应用）网页上使用，用来给HTML网页增加动态功能。

d.Unity3D：Unity3D是由Unity Technologies开发的一个让玩家轻松创建诸如三维视频游戏、建筑可视化、实时三维动画等类型互动内容的多平台的综合型游戏开发工具，是一个全面整合的专业游戏引擎。

## 参考资料

【1】张海藩、牟永敏编著，软件工程导论（第6版），北京：清华大学出版社，2013

【2】姚晓光、田少煦、梁冰、陈泽伟、伊宁编著，游戏设计概论，北京：清华大学出版社，2018

【3】谌宝业、魏伟、伍建平编著，游戏专业概论，北京：清华大学出版社，2018

【4】总体设计书模板：https://wenku.baidu.com/view/b06aab6327d3240c8447ef23.html

【5】《G15项目介绍书》、《G15项目计划书》、《G15项目可行性分析报告》、《G15项目需求分析报告》

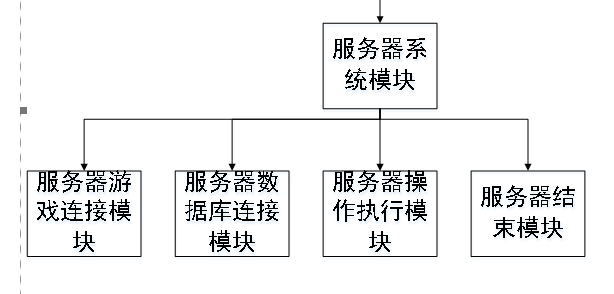
# 系统的结构



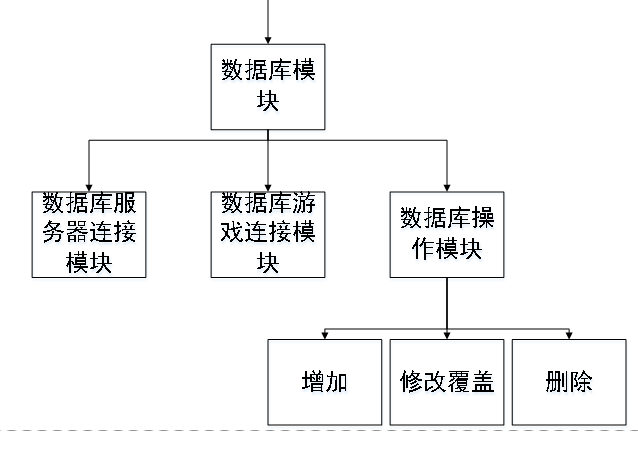
（以下是把图放大后的截图）

总模块

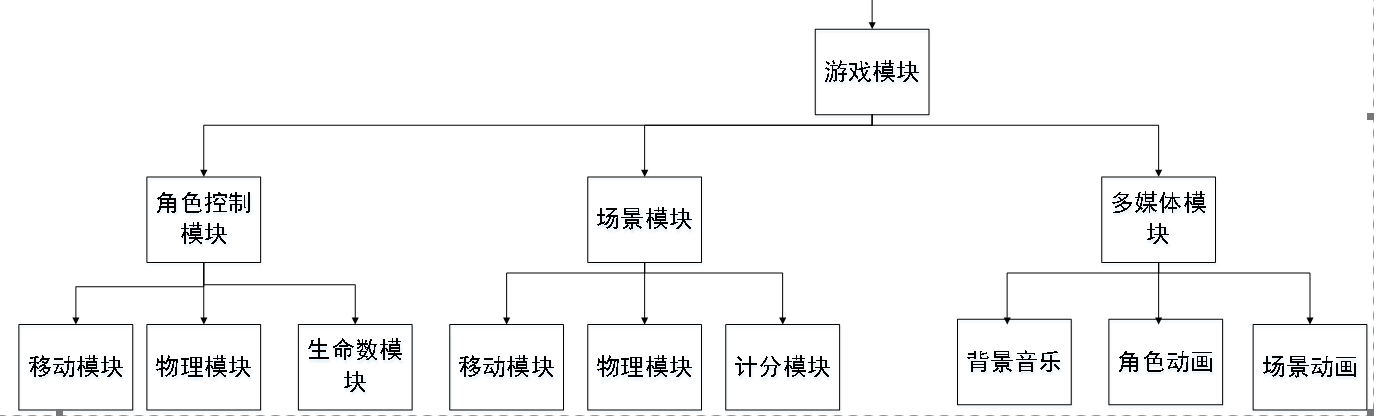
服务器系统模块部分：



数据库系统模块部分：



游戏模块部分：



# 服务器系统模块设计说明

## 模块描述

为了实现在网页上体验本游戏，我们需要有一台服务器来连接玩家，因此需要有一个服务期控制系统来实现相关功能。

## 功能

能够控制游戏本体与服务器的连接和断开，并将相关的数据在两者之间进行传输。

## 性能

至少能支持50人同时在线。

## 输入项

开始游戏时：打开该网页

结束游戏时：断开连接的操作，如断网或关闭网页

## 输出项

开始游戏时：连接游戏，并将游戏的开始界面显示出来

结束游戏时：断开游戏与服务器的连接

## 设计方法（算法）

选择算法

## 流程逻辑

If 打开该网页

Then 将游戏与服务器连接

End if

If 收到关闭的操作指令

Then 断开服务器与游戏的连接

End if

## 接口

开始时

当打开网页时

该模块发出打开游戏界面的指令

结束时：

当收到断开连接的指令

该模块断开连接

## 存储分配

待定

## 注释设计

暂无

## 限制条件

预算有限，我们用到的服务器是阿里云有学生优惠的，性能与企业版的相比会有所不如。

## 测试计划

测试是否能够正确连接与断开服务器和游戏

## 尚未解决的问题

游戏放到服务器上面无法正常打开

# 数据库模块设计说明

## 模块描述

用于存放游戏里会用到的相关数据，并将数据库与其他部分相连。

## 功能

增加、修改、删除数据

连接服务器与数据库

连接游戏与数据库

## 性能

至少能够存放5条游戏成绩数据

## 输入项

登录界面或游戏中对数据的改动指令

## 输出项

数据库中的数据变更

## 设计方法（算法）

// Learn cc.Class:

// - [Chinese] https://docs.cocos.com/creator/manual/zh/scripting/class.html

// - [English] http://docs.cocos2d-x.org/creator/manual/en/scripting/class.html

// Learn Attribute:

// - [Chinese] https://docs.cocos.com/creator/manual/zh/scripting/reference/attributes.html

// - [English] http://docs.cocos2d-x.org/creator/manual/en/scripting/reference/attributes.html

// Learn life-cycle callbacks:

// - [Chinese] https://docs.cocos.com/creator/manual/zh/scripting/life-cycle-callbacks.html

// - [English] https://www.cocos2d-x.org/docs/creator/manual/en/scripting/life-cycle-callbacks.html

cc.Class({

extends: cc.Component,

properties: {

bestRunScore: 0,

secondRunScore: 0,

thirdRunScore: 0,

fourthRunScore: 0,

fifthRunScore: 0,

sixthRunScore:0,

date:String,

date2:String,

date3:String,

date4:String,

date5:String,

date6:String,

},

onLoad: function () {

cc.game.addPersistRootNode(this.node);

var bestRunScore = cc.sys.localStorage.getItem("bestRunScore");

var secondRunScore = cc.sys.localStorage.getItem("secondRunScore");

var thirdRunScore = cc.sys.localStorage.getItem("thirdRunScore");

var fourthRunScore = cc.sys.localStorage.getItem("fourthRunScore");

var fifthRunScore = cc.sys.localStorage.getItem("fifthRunScore");

var date = cc.sys.localStorage.getItem("date");

var date2 = cc.sys.localStorage.getItem("date2");

var date3 = cc.sys.localStorage.getItem("date3");

var date4 = cc.sys.localStorage.getItem("date4");

var date5 = cc.sys.localStorage.getItem("date5");

if(bestRunScore){

this.bestRunScore = bestRunScore;

}

if(secondRunScore){

this.secondRunScore = secondRunScore;

}

if(thirdRunScore){

this.thirdRunScore = thirdRunScore;

}

if(fourthRunScore){

this.fourthRunScore = fourthRunScore;

}

if(fifthRunScore){

this.fifthRunScore = fifthRunScore;

}

if(date){

this.date = date;

}

if(date2){

this.date2 = date2;

}

if(date3){

this.date3 = date3;

}

if(date4){

this.date4 = date4;

}

if(date5){

this.date5 = date5;

}

if(scorecount){

this.scorecount = parseInt(scorecount) ;

}

},

updateRunScore: function(score){

var DATE = new Date();

var array=new Array(this.bestRunScore, this.secondRunScore, this.thirdRunScore, this.fourthRunScore,this.fifthRunScore);

var timearray=new Array(this.date,this.date2,this.date3,this.date4,this.date5);

for(var i=0;i<array.length;i++){

if(array[i]==0){

array[i]=score;

timearray[i]=DATE.toDateString()+DATE.toLocaleTimeString();

break;

}

if(score>this.fifthRunScore&&this.fifthRunScore!=0){

this.sixthRunScore=score

this.date6=DATE.toDateString()+DATE.toLocaleTimeString();

array=new Array(this.bestRunScore, this.secondRunScore, this.thirdRunScore, this.fourthRunScore,this.fifthRunScore,this.sixthRunScore);

timearray=new Array(this.date,this.date2,this.date3,this.date4,this.date5,this.date6);

}

}

for(var i=0;i<array.length-1;i++){

var higherscore=array[i];

var time=timearray[i]

for(var a=i+1;a<array.length;a++){

if(array[a]>=higherscore){

array[i]=array[a];

array[a]=higherscore;

higherscore=array[i];

timearray[i]=timearray[a];

timearray[a]=time;

time=timearray[i];

}

}

}

this.bestRunScore=array[0];

this.secondRunScore=array[1];

this.thirdRunScore=array[2];

this.fourthRunScore=array[3];

this.fifthRunScore=array[4];

this.date=timearray[0];

this.date2=timearray[1];

this.date3=timearray[2];

this.date4=timearray[3];

this.date5=timearray[4];

this.save();

},

save(){

cc.sys.localStorage.setItem('bestRunScore', this.bestRunScore);

cc.sys.localStorage.setItem('secondRunScore', this.secondRunScore);

cc.sys.localStorage.setItem('thirdRunScore', this.thirdRunScore);

cc.sys.localStorage.setItem('fourthRunScore', this.fourthRunScore);

cc.sys.localStorage.setItem('fifthRunScore', this.fifthRunScore);

cc.sys.localStorage.setItem('date', this.date);

cc.sys.localStorage.setItem('date2', this.date2);

cc.sys.localStorage.setItem('date3', this.date3);

cc.sys.localStorage.setItem('date4', this.date4);

cc.sys.localStorage.setItem('date5', this.date5);

},

});

## 流程逻辑

Loop While有改动数据

将修改后的数据额保存到数据库

End loop

## 接口

当登陆界面或游戏中发出会影响数据的指令时，该模块对数据库中的数据进行修改

## 存储分配

大部分存储空间

## 注释设计

暂无

## 限制条件

数据库的大小和性能可能很一般。

## 测试计划

在游戏中进行一些会修改数据的操作，测试数据库中的数据是否能够进行同步变更

## 尚未解决的问题

只能在本地进行连接，无法用于网络。

# 角色控制模块设计说明

## 模块描述

可以对角色的相关信息进行控制

## 功能

能够控制人物的移动跳跃等

判定角色与特殊地形是否发生物理碰撞

## 性能

玩家的角色控制指令与角色的实际移动之间的延迟不超过0.1s

## 输入项

玩家的操作指令

## 输出项

角色进行移动

## 设计方法（算法）

cc.Class({

extends: cc.Component,

properties: {

speed: cc.v2(0, 0),

movespeed:400,

maxSpeed: cc.v2(2000, 2000),

gravity: -1000,

drag: 1000,

direction: 0,

jumpSpeed: 300,

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

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

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

this.collisionX = 0;

this.collisionY = 0;

this.prePosition = cc.v2();

this.preStep = cc.v2();

this.touchingNumber = 0;

//设置暂停界面可见性

this.stopScene.node.active=false;

},

onEnable: function () {

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

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

},

onDisable: function () {

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

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

},

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;

case cc.macro.KEY.escape:

cc.director.loadScene('开始界面');

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;

}

},

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) {

if(this.stopflag==true){

cc.director.pause();

}

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;

}

}

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;

}

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;

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

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

},

});

## 流程逻辑

While 发出指令

{

根据指令移动角色

While 角色受到伤害

{

显示受伤后的生命值

}

}

## 接口

当收到控制角色移动的指令时，角色进行相关操作

## 存储分配

无

## 注释设计

暂无

## 限制条件

技术有限，我们只打算让角色具备基本的左右移动及跳跃的控制方式

## 测试计划

我们会测试我们键入的操作是否能够让角色进行正确的移动

## 尚未解决的问题

无

# 场景模块设计说明

## 模块描述

该模块可以显示并控制地图场景上的各个元素

## 功能

执行角色与场景中的陷阱和得分点的物理碰撞

让场景跟着角色的移动进行变换

让角色的分数实时显示到屏幕上

## 性能

可以准确的判断物理碰撞，场景的变换要跟着角色的移动实时变更，能准确显示分数

## 输入项

角色的移动信息

## 输出项

场景的变化

分数的变化

## 设计方法（算法）

计分：

cc.Class({

extends: cc.Component,

properties: {

speed: cc.v2(0, 0),

movespeed:400,

maxSpeed: cc.v2(2000, 2000),

gravity: -1000,

drag: 1000,

direction: 0,

jumpSpeed: 300,

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

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

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

this.collisionX = 0;

this.collisionY = 0;

this.prePosition = cc.v2();

this.preStep = cc.v2();

this.touchingNumber = 0;

//设置暂停界面可见性

this.stopScene.node.active=false;

},

onEnable: function () {

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

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

},

onDisable: function () {

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

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

},

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;

case cc.macro.KEY.escape:

cc.director.loadScene('开始界面');

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;

}

},

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) {

if(this.stopflag==true){

cc.director.pause();

}

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;

}

}

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;

}

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;

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

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

},

});

场景移动：

cc.Class({

extends: cc.Component,

properties: {

speed: cc.v2(0, 0),

movespeed:400,

maxSpeed: cc.v2(2000, 2000),

gravity: -1000,

drag: 1000,

direction: 0,

jumpSpeed: 300,

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

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

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

this.collisionX = 0;

this.collisionY = 0;

this.prePosition = cc.v2();

this.preStep = cc.v2();

this.touchingNumber = 0;

//设置暂停界面可见性

this.stopScene.node.active=false;

},

onEnable: function () {

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

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

},

onDisable: function () {

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

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

},

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;

case cc.macro.KEY.escape:

cc.director.loadScene('开始界面');

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;

}

},

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) {

if(this.stopflag==true){

cc.director.pause();

}

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;

}

}

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;

}

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;

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

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

},

});

场景加载：

cc.Class({

extends: cc.Component,

properties: {

speed: cc.v2(0, 0),

movespeed:400,

maxSpeed: cc.v2(2000, 2000),

gravity: -1000,

drag: 1000,

direction: 0,

jumpSpeed: 300,

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

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

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

this.collisionX = 0;

this.collisionY = 0;

this.prePosition = cc.v2();

this.preStep = cc.v2();

this.touchingNumber = 0;

//设置暂停界面可见性

this.stopScene.node.active=false;

},

onEnable: function () {

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

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

},

onDisable: function () {

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

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

},

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;

case cc.macro.KEY.escape:

cc.director.loadScene('开始界面');

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;

}

},

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) {

if(this.stopflag==true){

cc.director.pause();

}

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;

}

}

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;

}

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;

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

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

},

});

## 流程逻辑

While 角色移动

{

场景跟着变换

While 分数增加

{

显示增加后的分数

}

}

## 接口

当角色移动时，场景跟着变换

当角色符合加分条件时，分数增加并显示

## 存储分配

无

## 注释设计

暂无

## 限制条件

场景中的得分道具是固定的，所以可能会有相同的分数出现。

## 测试计划

测试人物移动时场景是否会实时变换，分数是否会实时变更并显示，角色与地图的物理碰撞是否准确。

## 尚未解决的问题

无

# 多媒体模块

## 模块描述

该模块为游戏提供角色动画以及场景动画。

## 功能

在用户进行游戏时，提供游戏角色不同动作的动画效果以及场景变化时的动画效果。

## 性能

该模块为游戏提供动画效果，增加用户的游戏体验。

## 输入项

无。

## 输出项

进入游戏时，系统提供背景音乐和动画效果。

## 设计方法（算法）

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

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

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

this.collisionX = 0;

this.collisionY = 0;

this.prePosition = cc.v2();

this.preStep = cc.v2();

this.touchingNumber = 0;

//设置暂停界面可见性

this.stopScene.node.active=false;

},

onEnable: function () {

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

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

},

onDisable: function () {

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

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

},

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;

case cc.macro.KEY.escape:

cc.director.loadScene('开始界面');

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;

}

},

## 流程逻辑

If 进入游戏

Then 系统执行背景音乐、角色动画、场景动画

End if

## 接口

当用户开始进行游戏时，系统会根据不同的游戏进度执行角色动画、场景动画。

## 存储分配

无

## 注释设计

暂无

## 限制条件

该模块只有在用户进行游戏时才开始运行。由于技术原因，动画效果并不能十分完美。

## 测试计划

开始进行游戏后，测试人物动画是否可以正常播放。

## 尚未解决的问题

动画比较僵硬。