**弹幕游戏引擎详细设计文档**

3010218064 yuxiaobhcs

3010218066 tjumyk

3010218073 谢利娟

2013年7月15日

目录

[一、模型详细设计 4](#_Toc361666490)

[Model包 4](#_Toc361666491)

[StageGroup 4](#_Toc361666492)

[Stage 4](#_Toc361666493)

[GameModel 5](#_Toc361666494)

[BaseObject implements Poolable 7](#_Toc361666495)

[Background extends BaseObject 10](#_Toc361666496)

[Bullet extends BaseObject implements Poolable 10](#_Toc361666497)

[ImageClass 11](#_Toc361666498)

[Enemybase extends BaseObject 12](#_Toc361666499)

[Enemy extends Enemybase 13](#_Toc361666500)

[Boss extends Enemybase 15](#_Toc361666501)

[Bosscard 16](#_Toc361666502)

[EnemyDeath extends BaseObject 17](#_Toc361666503)

[Bubble extends BaseObject 18](#_Toc361666504)

[Item extends BaseObject 19](#_Toc361666505)

[Laser extends BaseObject 20](#_Toc361666506)

[ParticleObject extends BaseObject 22](#_Toc361666507)

[Grazer extends ParticleObject 23](#_Toc361666508)

[PlayerDeath extends ParticleObject 23](#_Toc361666509)

[PlayerBullet extends BaseObject 23](#_Toc361666510)

[PlayerBulletStraight extends PlayerBullet 24](#_Toc361666511)

[PlayerBulletHide extends PlayerBullet 24](#_Toc361666512)

[PlayerBulletTrail extends PlayerBullet 25](#_Toc361666513)

[PlayerClass extends BaseObject 26](#_Toc361666514)

[model.player包 28](#_Toc361666515)

[PlayerReimu extends PlayerClass 28](#_Toc361666516)

[ReimuBulletRed extends PlayerBulletStraight 28](#_Toc361666517)

[ReimuBulletBlue extends PlayerBulletTrail 29](#_Toc361666518)

[ReimuSp extends PlayerBulletTrail 29](#_Toc361666519)

[ReimuSpEffect extends PlayerBulletHide 30](#_Toc361666520)

[状态流程设计 30](#_Toc361666521)

[二、脚本引擎详细设计 33](#_Toc361666522)

[概述 33](#_Toc361666523)

[脚本引擎结构设计 33](#_Toc361666524)

[1. 表达式解析与执行模块 33](#_Toc361666525)

[2. Task逻辑控制模块 33](#_Toc361666526)

[3. XML脚本解析模块 33](#_Toc361666527)

[详细设计 34](#_Toc361666528)

[1. 表达式解析 34](#_Toc361666529)

[2. Task逻辑块 34](#_Toc361666530)

[3. XML解析 35](#_Toc361666531)

# 一、模型详细设计

## Model包

\*注：在类的说明中，get/set函数的说明被省略

### StageGroup

**Description:**

一个关卡组，代表一个难度模式下的所有游戏关卡

**Data field:**

title:String 关卡组名

stage\_group:ArrayList<Stage> 组内关卡列表

**Method:**

public StageGroup(String title) 构造函数

title:关卡组名称

public void addStage(Stage stage) 添加一个关卡到stage\_group最后

stage:添加的关卡

public void addStage(String stagename, int life, int power, int faith) 添加一个关卡

参数意义见Stage的构造函数

public Stage getStagebyId(int id)

id:根据序号获取列表中的Stage

#### 类图：



### Stage

**Description:**

一个关卡，游戏进行的基本场景单元

**Data field:**

title:String 关卡名

life,power,faith:int 以该关卡为开始游戏的第一关时初始残机，火力，信仰

task:Task 关卡Task

**Method:**

public Stage(String title, int life, int power, int faith) 构造函数

title:关卡名称

life:初始残机

power:初始火力

faith:初始信仰

#### 类图：



### GameModel

**Description:**

保存控制游戏进行时的所有关键信息的GM类

**Data field:**

screenWidth,screenHeight:int 屏幕的实际宽高

worldWidth,worldHeight:int 世界坐标的宽高

gameStage:ArrayList<StageGroup> 游戏包含的关卡组

currentStage:Stage 当前关卡

isPratice:boolean 当前是否为练习模式

pauseFlag:boolean 是否在暂停状态

quitFlag:boolean 是否在退出状态

bossWait:boolean 是否在等待Boss击破状态

player:PlayerClass 自机

diffcultid,stageid:int 当前难度，关卡序号

objList:ArrayList<BaseObject> 所有单位的列表

life:int 当前游戏状态剩余残机数

power:int 当前游戏状态火力值

faith:int 当前游戏状态信仰值

chip:int 当前游戏状态生命碎片个数(5个碎片可换一残机)

graze:int 当前游戏状态擦弹数目

score:int 当前游戏状态游戏分数

blockSpell:boolean 当前是否禁止自机使用符卡(即B弹)

chipBonus:boolean 当前是否有生命碎片奖励(这个忽略，根本没看到相关脚本代码)

gameBG:Background 当前背景

keyShoot:boolean 当前帧玩家是否进行了射击操作(或自机为射击状态)

keySpell:boolean 当前帧玩家是否进行了符卡操作

keySlow:boolean 当前帧玩家是否进行了低速操作(或自机为低速状态)

keyMove:boolean 当前帧玩家是否进行了移动操作

playerMoveAngel:double 当前帧玩家移动的方向角度

bulletPool:Pool<Bullet> Bullet类对象池

**Method:**

public static GameModel getInstance() 获取单例

public void init() 初始化stagegroup及stage

public void frame() 每帧的逻辑运算，包括所有obj的frame()，Movement()，碰撞检测

private GeometryBase getGeometry(BaseObject obj) 获取用于碰撞检测的几何外形对象

obj:用于碰撞的对象

public void startStage(int diffcultid, int stageid) 开始关卡

diffcultid: 关卡组组中难度序号

stageid: 关卡组中关卡序号

public void addStageGroup(String sgname) 添加关卡组

sgname:关卡组名

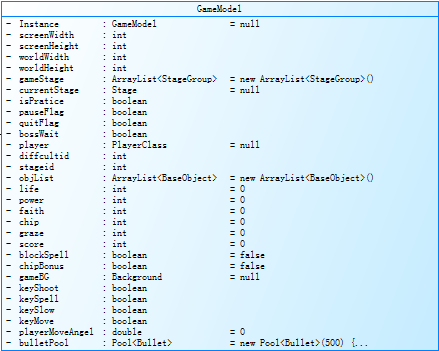
public void nextStage() 进入下一关，如当前为最后一关则通关

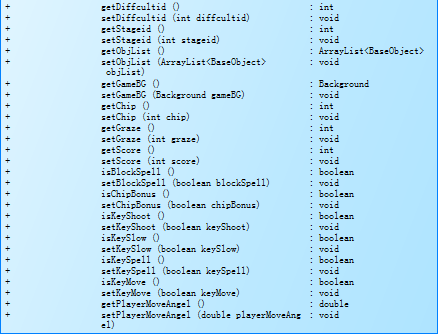
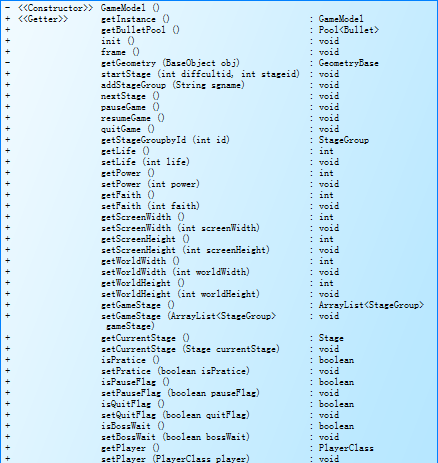
public void pauseGame() 游戏暂停

public void resumeGame() 游戏恢复

public void quitGame() 游戏退出

#### 类图：





### BaseObject implements Poolable

**Description:**

所有单位的基类，定义单位基本属性

**Data field:**

x:double 单位的X坐标，默认为0

y:double 单位的Y坐标，默认为0

rot:double 单位的朝向，默认为0(0为正右，顺时针增加)

vx:double 单位的x坐标的每帧的自增量，默认为0

vy:double 单位的y坐标的每帧的自增量，默认为0

omega:double 单位的朝向的每帧的自增量，默认为0

group:int 单位所属的碰撞组，默认为GROUP\_GHOST

/\*\*

在每帧，做如下的碰撞检查：

CollisionCheck(GROUP\_PLAYER,GROUP\_ENEMY\_BULLET)

CollisionCheck(GROUP\_PLAYER,GROUP\_ENEMY)

CollisionCheck(GROUP\_PLAYER,GROUP\_INDES)

CollisionCheck(GROUP\_ENEMY,GROUP\_PLAYER\_BULLET)

CollisionCheck(GROUP\_ITEM,GROUP\_PLAYER)

\*\*/

layer:double 单位的图层，默认为0 (越大的渲染越靠后，不过现在没有实现这个机制)

a:double 单位横向碰撞半径，默认为0

b:double 单位纵向碰撞半径，默认为0

rect:boolean 单位的碰撞判定是否为矩形，默认为false

timer:int 计数器，在单位被创建时为0，以后每帧+1

ani:int 单位的动画计数器。和timer类似，但是专门用于动画的渲染

status:int 单位的状态为Normal,Del,Kill之一，默认为Normal

img:String 单位的显示图像

hscale:double 单位的横向拉伸率，只影响图像大小，不影响碰撞，默认为1

vscale:double单位的纵向拉伸率，只影响图像大小，不影响碰撞，默认为1

bound:boolean 单位离开显示范围之后是否被自动清除,默认为true

colli:boolean 单位是否参与碰撞检测，默认为true

hide:boolean是否隐藏单位，如果为true，则跳过该单位的render的执行，默认为false

navi:boolean是否根据单位的运动方向自动计算并调整其朝向，默认为false

task:Task 该单位的task

**Method:**

public BaseObject() 构造函数

public void colli(BaseObject other) 碰撞函数

other:碰撞的对象

public void del() 清除函数

public void frame() 帧逻辑函数

public void init() 初始化函数

public void reset() 对象池重置函数

#### 类图：



### Background extends BaseObject

**Description:**

背景类

**Data field:**

alpha:double 背景图层的透明度

（剩下几个你自己写的我不懂）

initTask,frameTask,renderTask:Task

data:Map<String, Object>

**Method:**

public Background(boolean isScBg, String scriptPath) 构造函数

isScBg: 是否为符卡背景

scriptPath: Task脚本路径

public void init(boolean isScBg) 初始化函数

public void frame()

public void render()

#### 类图：



### Bullet extends BaseObject implements Poolable

**Description:**

敌机的子弹

**Data field:**

imgclass:ImageClass 针对Bullet的图像类

index:int 颜色序号，在GameUtil中定义了静态常量

img1,img2:String img1为子弹图像，img2为子弹产生时的产生点图像

stay:boolean 该子弹是否有创建时间，一般情况下为true

destroyable:boolean 该子弹是否是可破坏的，true则子弹归入GROUP\_ENEMY\_BULLET，false则子弹归入GROUP\_INDES

hasgrazed:boolean 该子弹是否被擦弹过，避免重复擦弹同一对象

**Method:**

public Bullet(String imgclassName, int index, double x, double y, double v,

double angle, boolean aim, double omiga, boolean stay,

boolean destroyable) 构造函数

参数见init()

public void init(String imgclassName, int index, double x, double y,

double v, double angle, boolean aim, double omiga, boolean stay,

boolean destroyable) 初始化函数

imgclassName: ImageClass类的名字

index:颜色序号

x:子弹初始x坐标

y:子弹初始y坐标

v:子弹初始速度

angle:子弹朝向偏移角度

aim:子弹是否根据自机偏移(只是偏移一个角度，不是指最后角度瞄准自机)

omega:子弹朝向自增量

stay:子弹是否有等待创建时间

destroyable:子弹是否可破坏

public void frame()

public void kill()

public void render()

public void reset() 对象池重置函数

#### 类图：



### ImageClass

**Description:**

描述Bullet类型的枚举类，定义了Bullet的图像，碰撞信息

**Data field:**

imgPrename:String 该种弹型的资源前缀名

interval:int 该种弹型不同颜色之间的颜色序号间隔（因为有的弹型16种颜色全有，有的弹型颜色少，interval保证资源名的正确）

a:int 该种弹型的横向碰撞半径

b:int 该种弹型的纵向碰撞半径

rect:boolean 该种弹型的碰撞判定是否为矩形

**Method:**

private ImageClass(String imgPrename, int interval, int a, int b,boolean rect) 构造函数

imgPrename: 该种弹型的资源前缀名

interval:颜色序号间隔

a,b,rect:碰撞检测信息

public void init(Bullet blt, int index) 初始化Bullet信息函数

blt:被初始化的Bullet对象

index:该Bullet的颜色序号

#### 类图：



### Enemybase extends BaseObject

**Description:**

敌机的基类

**Data field:**

protect:boolean 单位是否无敌

hp:double 单位的当前血量

maxhp:double 单位的最高血量

imgs:List<String> 单位的图像集，frame函数会根据单位当前运动状态选择图像

servants:List<BaseObject> 单位的Servant列表

master:BaseObject 单位的Master，Master死亡Servant也会死亡

drop:int[] 单位掉落的Item，下标0,1,2对应死亡时掉落的power,faith,point(火力,信仰,点数)

**Method:**

public Enemybase(double hp) 构造函数

public void init(double hp)

hp:敌机的血量

public void colli(BaseObject other)

other:碰撞的自机子弹

public void takeDamage(double dmg) 伤害计算函数

dmg:伤害值

public void frame()

public void del()

public void killServants(Enemybase master) 杀死所有servant

master: 操作的master对象

public void delServants(Enemybase master) 清除所有servant

master: 操作的master对象

#### 类图：



### Enemy extends Enemybase

**Description:**

普通敌机

**Static data field:**

Enemy\_aura\_tb:int[] 对象的光环图id，0表示没有光环，按style索引

Death\_ef\_tb:int[] 对象死亡时的Bubble图像id，按style索引

Enemy\_a\_tb:int[] 对象的横向碰撞半径，按style索引

Enemy\_b\_tb:int[] 对象的纵向碰撞半径，按style索引

**Data field:**

style:int 单位的类型序号（1—26之间）

clearBullet:boolean 单位死亡后是否触发BulletClear

autoDelete:boolean 单位是否出屏自动删除

aura:int 单位的光环图id

deathEf:int 单位的死亡Bubble图像id

aniIntv:int 单位的动画切换间隔

lr:int 单位的左右位移程度（小于0表示在左移，大于0表示右移，绝对值小于等于18）

**Method:**

public Enemy(int style, double hp, boolean clear\_bullet, boolean auto\_delete) 构造函数

public void init(int style, double hp, boolean clearBullet, boolean autoDelete) 初始化函数

style: 敌机的类型

hp: 生命值

clearBullet: 单位死亡后是否触发BulletClear

autoDelete: 单位是否出屏自动删除

public void frame()

public void render()

public void takeDamage(double dmg) 伤害计算函数

dmg: 伤害值

public void kill()

#### 类图：



### Boss extends Enemybase

**Description:**

Boss敌机

**Data field:**

aniIntv:int 单位的动画切换间隔

auraAlpha:int Boss背后光环的透明度

auraAlphaD:int Boss背后光环透明度的改变量

cardIndex:int 当前符卡的序号

cards:List<Bosscard> Boss的符卡列表

currentcard:Bosscard Boss的当前符卡

dmgFactor:double 伤害系数

lr:int单位的左右位移程度（小于0表示在左移，大于0表示右移，绝对值小于等于18）

scName:String 当前符卡名

bossName:String Boss名

\*PS:Boss被Kill时会判断是否还有符卡，若有则换到下张符卡，没有就真实死亡

**Method:**

public Boss(double x, double y, String name, List<Bosscard> cards) 构造函数

public void init(double x, double y, String name, List<Bosscard> cards)

x: 对象的初始x坐标

y: 对象的初始y坐标

name: Boss的名字

cards: Boss的符卡组

public void takeDamage(double dmg) 伤害计算函数

dmg: 伤害值

public void kill()

public void frame()

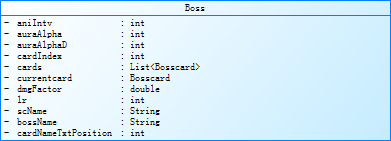
public void render()

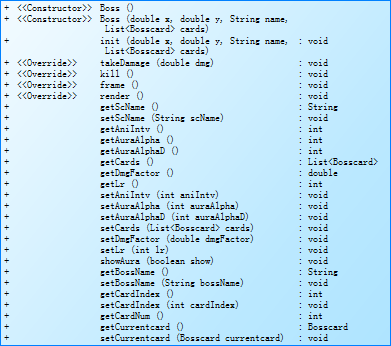
public void showAura(boolean show) 光环显示函数

show: 显示模式

public int getCardNum() 获取符卡数目

#### 类图：





### Bosscard

**Description:**

Boss符卡

**Data field:**

name:String 符卡的名字

t1:int 符卡第一阶段时间长度，此段伤害系数为0，即无敌

t2:int 符卡第二阶段时间长度，此段伤害系数渐变到1

t3:int 符卡第三阶段时间长度，此段伤害系数为1，此段过后符卡等于通过，Kill Boss一次

hp:double 符卡的血量

drop:int[] 符卡掉落的Item，下标0,1,2对应死亡时掉落的power,faith,point

task:Task 符卡所包含的的task

**Method:**

Bosscard(String name, int t1, int t2, int t3, double hp, int dpower, int dfaith, int dpoint) 构造函数

name: 符卡名

t1: 第一阶段时间

t2: 第二阶段时间

t3: 第三阶段时间

hp: 血量

dpower: 击破后掉落的power数

dfaith: 击破后掉落的faith数

dpoint: 击破后掉落的point数

#### 类图：



### EnemyDeath extends BaseObject

**Description:**

敌机单位死亡后死亡效果

**Method:**

public EnemyDeath(int index, double x, double y) 构造函数

public void init(int index, double x, double y) 初始化函数

index: 图像序号(1-4)

x: 对象的x坐标

y: 对象的y坐标

public void frame()

public void render()

#### 类图：



### Bubble extends BaseObject

**Description:**

气泡效果

**Data field:**

lifetime:int Bubble存在时间

size1:double 初始size

size2: double 结束size

**Method:**

public Bubble(String img, double x, double y, int lifeTime, double size1, double size2, double layer) 构造函数

public void init(String img, double x, double y, int lifeTime,

double size1, double size2, double layer)

img:图像

x: 对象的x坐标

y: 对象的y坐标

lifeTime: 存在时间

size1: 初始size

size2: 结束size

layer: 所在的渲染层

public void render()

public void frame()

#### 类图：



### Item extends BaseObject

**Description:**

掉落物

**Data field:**

imgup:String Item在屏外时的图像

attract:int Item收到自机的吸引系数

\*PS:model.item包中实现了全部的8种Item，他们的collect()函数不同

**Method:**

public Item(double x, double y, int t) 构造函数

public void init(double x, double y, int t)

x：对象的x坐标

y：对象的y坐标

t：Item的类型

public void render()

public void frame()

public void colli(BaseObject other)

public void collect() Item被收集时发生的事件

public static void dropItem(double x, double y, int[] drop) 产生Item掉落

x: x坐标

y: y坐标

drop: 掉落的各种Item的数目数组

public static void startChipBonus() 开启生命碎片奖励

public static void endChipBonus(double x, double y) 关闭生命碎片奖励，并掉落生命碎片

x,y：坐标信息

public static void playerInit() 自机的信息初始化(其实是初始化GM中关于自机的变量)

public static void playerMiss() 自机被击落的变量处理以及Item掉落

public static void playerGraze() 自机擦弹的变量处理

public static int pointRateFunc() 得分系数计算，与信仰有关

#### 类图：



### Laser extends BaseObject

**Description:**

直线激光

**Data field:**

index:int 激光的颜色序号

img1:String 激光头图像

img2:String 激光体图像

img3:String 激光尾图像

img4:String 激光发射点图像

l1:double 激光头长度

l2:double 激光体长度

l3:double 激光尾长度

w0:double 激光最大宽度

w:double 激光当前宽度

alpha:double 激光透明度

node:double 激光发射点大小系数

dw:double 激光宽度变化量

da:double 激光透明度变化量

counter:int 内部参数变化计数器

**Method:**

public Laser(int index, double x, double y, double rot, double l1,

double l2, double l3, double w, double node) 构造函数

public void init(int index, double x, double y, double rot, double l1,

double l2, double l3, double w, double node)

index: 颜色序号(1-8)

x,y: 坐标信息

rot: 激光朝向角度

l1,l2,l3: 激光的长度信息

w: 最大宽度

node: 发射点大小

public void frame()

public void render()

public void turnOn(int t, boolean mute) 激光开

t: 打开所用时间

mute: 是否静默

public void turnHalfOn(int t) 激光半开

t: 打开所用时间

public void turnOff(int t) 激光关

t: 关闭所用时间

#### 类图：



### ParticleObject extends BaseObject

**Description:**

有粒子特效开关的类的基类

**Data field:**

fireParticle:boolean 当前是否发射粒子特效(false则当前帧不渲染该单位粒子特效)

**Method:**

public void init() 初始化, fireParticle设为true。

#### 类图：



### Grazer extends ParticleObject

**Description:**

与自机同步移动，自机擦弹判定圆，擦弹特效类

**Data field:**

grazed:boolean 当前帧是否被擦弹

**Method:**

public void init()

public void frame()

public void render()

public void colli(BaseObject other)

#### 类图：



### PlayerDeath extends ParticleObject

**Description:**

自机死亡后效果类

**Method:**

public PlayerDeath(double x, double y) 构造函数

public void init(double x, double y)

x,y: 坐标信息

public void frame()

public void render()

#### 类图：



### PlayerBullet extends BaseObject

**Description:**

自机子弹基类

**Data field:**

dmg:double 自机子弹伤害

**Method:**

public void init(String img, double x, double y, double v, double angle, double dmg)

img: 图像

x,y: 坐标信息

v,angle: 运动速度和角度

dmg: 伤害值

#### 类图：



### PlayerBulletStraight extends PlayerBullet

**Description:**

直线自机子弹

**Method:**

public PlayerBulletStraight(String img, double x, double y, double v, double angle, double dmg)

public void init(String img, double x, double y, double v, double angle, double dmg)

img: 图像

x,y: 坐标信息

v,angle: 运动速度和角度

dmg: 伤害值

#### 类图：



### PlayerBulletHide extends PlayerBullet

**Description:**

隐形自机子弹

**Data field:**

delay:int 延迟时间(Timer到达delay时间后，隐形子弹变为有伤害效果，此前无伤害效果)

**Method:**

public PlayerBulletHide(double a, double b, double x, double y, double v,

double angle, double dmg, int delay) 构造函数

public void init(double a, double b, double x, double y, double v,

double angle, double dmg, int delay)

a,b: 碰撞信息

x,y: 坐标信息

v,angle: 运动速度和角度

dmg: 伤害值

delay: 延迟时间

public void frame()

#### 类图：



### PlayerBulletTrail extends PlayerBullet

**Description:**

追踪自机子弹

**Data field:**

target:BaseObject 该追踪弹的目标Obj

trail:double 调整追踪弹追踪模式的参数，一般在3000至6000

v:double 追踪弹的速度

**Method:**

public PlayerBulletTrail(String img, double x, double y, double v,

double angle, BaseObject target, double trail, double dmg) 构造函数

public void init(String img, double x, double y, double v, double angle,

BaseObject target, double trail, double dmg)

img: 图像

x,y: 坐标信息

v,angle: 运动速度和角度

target: 该追踪弹的目标

trail: 调整追踪弹追踪模式的参数，一般在3000至6000

dmg: 伤害值

public void frame()

#### 类图：



### PlayerClass extends BaseObject

**Description:**

自机基类

**Data field:**

imgs:List<String> 图像列表

supportx:double 支援物基础X坐标

supporty:double 支援物基础Y坐标

hspeed:double 高速速度

lspeed:double 低速速度

dx:double 上一帧X位移

dy:double 上一帧Y位移

collectLine:double Item收集线(当越过此线收集所有Item)

slow:int 速度模式标记，1为低速，0为高速

lr:int 左右位移程度

lh:double 高低速变换程度

fire:double 火力展开程度(无需关心的一个量)

lock:boolean 是否被锁定(无需关心的用来作弊的量)

dialog:boolean 当前是否在剧情

nextshoot:int 下一次允许射击还剩多少帧

nextspell:int 下一次允许使用符卡还剩多少帧

death:int 死亡计数，death为0时为正常状态，大于0时表示在执行死亡后动作

protect:int 剩余无敌时间

grazer:Grazer 自机擦弹判定类

support:int 支援物等级(即火力等级)

hasslist:boolean 是否拥有可用的slist数组

slist:double[][][] 计算支援物偏移量的数组

sp:double[][] 支援物偏移量数组

target:BaseObject 供追踪弹追踪的目标

**Method:**

public PlayerClass() 构造函数

public void init() 初始化函数

public void frame()

public void shoot() 自机射击动作函数

public void spell() 自机符卡动作函数

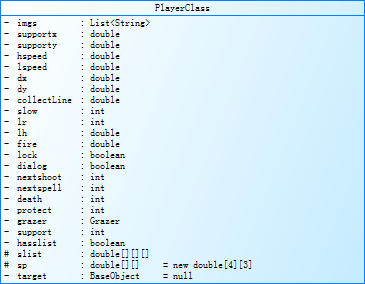
public double[] mixTable(double x, double[] t1) 坐标混合函数

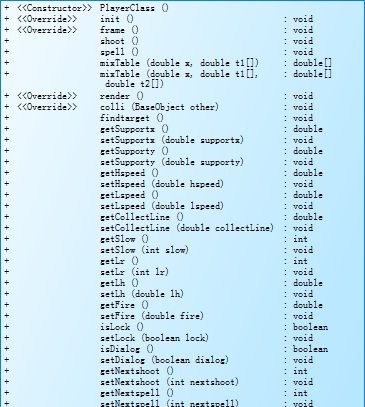
public void render()

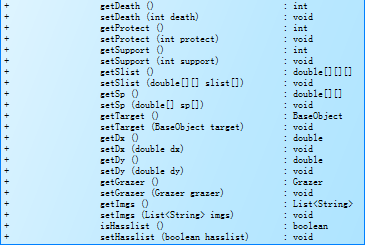
public void colli(BaseObject other)

public void findtarget() 目标搜寻函数

#### 类图：







## model.player包

### PlayerReimu extends PlayerClass

**Description:**

自机“博丽灵梦”

**Method:**

public PlayerReimu()

public void init()

public void shoot() 灵梦射击动作函数

public void spell() 灵梦符卡动作函数

public void render()

#### 类图：



### ReimuBulletRed extends PlayerBulletStraight

**Description:**

灵梦的红色直线弹

**Method:**

public ReimuBulletRed(String img, double x, double y, double v,

double angle, double dmg) 构造函数

public void render()

#### 类图：



### ReimuBulletBlue extends PlayerBulletTrail

**Description:**

灵梦的蓝色追踪弹

**Method:**

public ReimuBulletBlue(String img, double x, double y, double v,

double angle, BaseObject target, double trail, double dmg) 构造函数

public void render()

#### 类图：



### ReimuSp extends PlayerBulletTrail

**Description:**

灵梦的符卡

**Method:**

public ReimuSp(String img, double x, double y, double v, double angle,

BaseObject target, double trail, double dmg) 构造函数

public void kill()

public void render()

#### 类图：



### ReimuSpEffect extends PlayerBulletHide

**Description:**

灵梦符卡的后续隐形弹

**Data field:**

life:int 持续时间

**Method:**

public ReimuSpEffect(double a, double b, double x, double y, double v,

double angle, double dmg, int delay, int life) 构造函数

public void init(double a, double b, double x, double y, double v,

double angle, double dmg, int delay, int life)

a,b: 碰撞信息

x,y: 坐标信息

v,angle: 运动速度和角度

dmg: 伤害值

delay: 延迟时间

life: 存在时间

public void frame()

#### 类图：



## 状态流程设计

游戏流程的数据模型部分由GameModel类控制，GameModel的状态决定游戏状态

#### GameModel状态图



在关卡内时，游戏按帧进行，GameModel每帧与游戏中的所有BaseObject交互，更新所有BaseObject对象内的数据，维护游戏的进行。

#### GameModel与BaseObject帧内活动时序图



#### 样例代码：

currentStage.getTask().act();

**if**(**this**.gameBG != **null**)

**this**.gameBG.frame();

**for** (**int** i = 0; i < getObjList().size(); i++) {

BaseObject obj = objList.get(i);

**if** (!GameUtil.*IsValid*(obj)) {

**if** (obj **instanceof** Bullet) {

getBulletPool().free((Bullet) obj);

}

getObjList().remove(i);

i--;

} **else** {

obj.frame();

}

}

**for** (BaseObject obj : getObjList()) {

GameUtil.*Movement*(obj);

}

// 碰撞检测

**for** (**int** i = 0; i < getObjList().size(); i++) {

BaseObject obj1 = getObjList().get(i);

**if** (!obj1.isColli())

**continue**;

**for** (**int** j = 0; j < getObjList().size(); j++) {

BaseObject obj2 = getObjList().get(j);

**if** (obj1 == obj2 || !obj2.isColli())

**continue**;

**if** ((obj1.getGroup() == GameUtil.*GROUP\_PLAYER* && obj2

.getGroup() == GameUtil.*GROUP\_ENEMY\_BULLET*)

|| (obj1.getGroup() == GameUtil.*GROUP\_PLAYER* && obj2

.getGroup() == GameUtil.*GROUP\_ENEMY*)

|| (obj1.getGroup() == GameUtil.*GROUP\_PLAYER* && obj2

.getGroup() == GameUtil.*GROUP\_INDES*)

|| (obj1.getGroup() == GameUtil.*GROUP\_ENEMY* && obj2

.getGroup() == GameUtil.*GROUP\_PLAYER\_BULLET*)

|| (obj1.getGroup() == GameUtil.*GROUP\_ITEM* && obj2

.getGroup() == GameUtil.*GROUP\_PLAYER*)) {

**if** (CollisionCheck.*check*(getGeometry(obj1),

getGeometry(obj2))) {

obj1.colli(obj2);

}

}

}

}

# 二、脚本引擎详细设计

## **概述**

脚本引擎是游戏中重要的组件，可用于动态管理资源和逻辑，可实现游戏内容的快速编写与维护。脚本化的结构设计，可将游戏引擎本身与游戏内容实现一定程度的分离，从而让游戏开发过程更加便捷可靠。

本弹幕游戏引擎中也包含了一个基于XML和类EL表达式的脚本引擎，可以从文本文件读入脚本、解析脚本，并执行脚本。该脚本引擎可以实现表达式的解析与执行、脚本逻辑控制、执行状态控制、数据绑定以及脚本列表的读入、解析和管理。

本游戏引擎使用Java语言开发，所以本脚本引擎也是使用Java实现。

## 脚本引擎结构设计

### 表达式解析与执行模块

表达式是脚本引擎的最基本的模块，它用于处理脚本中最小的任务单元。表达式可以执行基本的算术运算，新建局部变量，遍历数组和Map。另外，它还能与Java进行互操作，例如调用Java对象的函数，访问Java对象的public数据域，以及调用Java类的静态函数，并且，脚本的执行结果可以返回到Java代码中。

### Task逻辑控制模块

一个完整的脚本由多个表达式组成，表达式之间存在一定的逻辑关系。本脚本引擎用Java类封装了基本的逻辑结构，包括：顺序结构、选择分支结构、循环结构，以及两种执行体：表达式、等待N帧，并且以上元素能够互相组合，从而形成一个复杂的Task任务。每个Task能够逐帧运行，即每次执行仅仅运行一帧内的动作，并记录当前的状态，在下一次执行Task时，从上次中断的位置继续运行。这中执行方式适合基于帧刷新的游戏系统。

### XML脚本解析模块

Task模块对应一系列的Java类，在使用过程中不是很方便，所以我们需要用脚本文件来编写和读入Task的内容。本脚本引擎支持从XML格式的文本文件中读入Task，其中不同的标签对应于不同的Task单元，标签之间互相嵌套，从而使Task的逻辑结构更清晰地表现出来。

另外，一个游戏对象可以用Task脚本实现一些动态的动作，并且可能需要在不同地方使用多个Task。因此，本脚本引擎也支持TaskList脚本的读取，TaskList脚本将多个Task脚本放在一起，从而避免脚本文件碎片化。

## 详细设计

### 表达式解析

表达式解析采用了Apache的开源项目Jexl，它支持类EL表达式的解析与执行，能够实现该模块所需要的所有功能。

表达式解析时，需要指定一个上下文环境context，其中使用了一个自定义的Map结构存储了context信息。该Map支持上下文的相互嵌套。从而可以实现子节点访问父节点的上下文信息。

### Task逻辑块

* 1. Sequence：顺序执行块，其中包含的子逻辑块将顺序执行。
  2. Parallel：并行执行块，其中包含的子逻辑块将并行执行。
  3. IfElse：分支（选择）执行块，其中可包含一个或两个子逻辑块，并包含一个用于执行判断的表达式。
  4. Loop：循环执行块，其中包含一个可循环若干次执行的逻辑块。
  5. Wait：等待执行块，用于等待若干帧。
  6. Script：脚本执行块，其中包含一段脚本，使用表达式解析模块进行执行。
  7. Empty：空执行块。

由于游戏逻辑结构的需要，Task执行时，每次只执行一帧的内容，当一帧被消耗掉后，将中止执行，立即返回。而当Task被再次执行时，会从上次中断的地方继续运行。

在Java代码中，可以通过task.setBinding(name,value)来实现对每个Task中context数据的绑定。从而让叶子节点的脚本块能操作这些数据。

### XML解析

Task逻辑块可相互组合，形成复杂的树状结构，其结构用XML表示，即写成XML文件的形式。

样例如下：

|  |
| --- |
| <!-- 顺序执行块 -->  <seq>  <!-- 循环执行块 -->  <loop count=**"5"**>  <!-- 脚本执行块 -->  <cmd>  var i = 0;  game.nextStage();  </cmd>  </loop>  </seq> |

为了便于设定上下文中的数据，XML 中还支持<var>标签，用于在当前节点的上下文中声明一个新的变量，供本节点及其子节点访问和修改。

为了实现脚本的可重用性，防止单个脚本过于繁杂，XML中支持<script>标签来引用别的XML文件。

在本游戏中，游戏角色的移动任务很普遍，所以XML支持<move>标签来快速实现一个角色移动任务。

游戏角色的逻辑特征往往包含多个方面，所以不是一个Task脚本可以定义的，因此多个上述XML脚本可以被置于<tasklist>标签中，从而形成一个完整的XML文本文件。

样例如下：

|  |
| --- |
| <!-- Task列表 -->  <tasklist>  <!-- 一个Task的逻辑结构 -->  <task name=**"task1"**>  <!-- 脚本执行块 -->  <cmd>  **var i = 0;**  </cmd>  </task>    <!-- 另一个Task的逻辑结构 -->  <task name=**"task2"**>  <!-- 脚本执行块 -->  <cmd>  **var i = 0;**  </cmd>  </task>  </tasklist> |