目录

[战斗场景系统 2](#_Toc91758415)

[1.综述 2](#_Toc91758416)

[2.用户需求 2](#_Toc91758417)

[3.系统总体结构 2](#_Toc91758418)

[4.类图的设计以及说明 3](#_Toc91758419)

[4.1 战斗场景控制器 3](#_Toc91758420)

[4.2关卡类 4](#_Toc91758421)

[4.3 规则类 6](#_Toc91758422)

[4.4 费用控制器 6](#_Toc91758423)

[4.5 卡片建造器 7](#_Toc91758424)

[4.6 技能控制器 7](#_Toc91758425)

[4.7 游戏进度控制器 7](#_Toc91758426)

[4.8 格子 7](#_Toc91758427)

[4.9 单位 7](#_Toc91758428)

[4.10 键位控制接口 7](#_Toc91758429)

# 战斗场景系统

## 1.综述

战斗场景即玩家在进入关卡之后正式进行游玩的部分，玩家可以通过合理的搭配卡片组成坚固的防线来防守敌人的进攻，直到最终胜利或者失败，需要使用Unity中单独的一个Scene实现。

## 2.用户需求

A：关卡设计上希望能设计几套固定的出怪模版，然后可以自定义几个不同的模式，并且支持上述出怪模版、游戏模式可以任意两两组合进游戏里。拿原版举例子就是，这里先写了一个曲奇岛的出怪模版和魔塔130的出怪模版，游戏模式写了普通通关战模式和争分夺秒模式，最终要系统不仅能做到普通通关战模式的曲奇岛出怪模版、和争分夺秒模式的魔塔130出怪模版，也能做到争分夺秒模式的曲奇岛出怪模版和普通通关战模式的魔塔130出怪模版。

B：希望自定义的模式能够有极大设计自由度，能非常舒服得制订规则，比如说失败条件不一定像传统的老鼠进家或者人物死亡，可以改为别的，相应的判定也用代码实现。胜利条件也可以改成其它，换言之在符合开闭原则的基础上只通过增加新的类就能实现PVZ里所有支线益智小游戏的成功与失败条件。

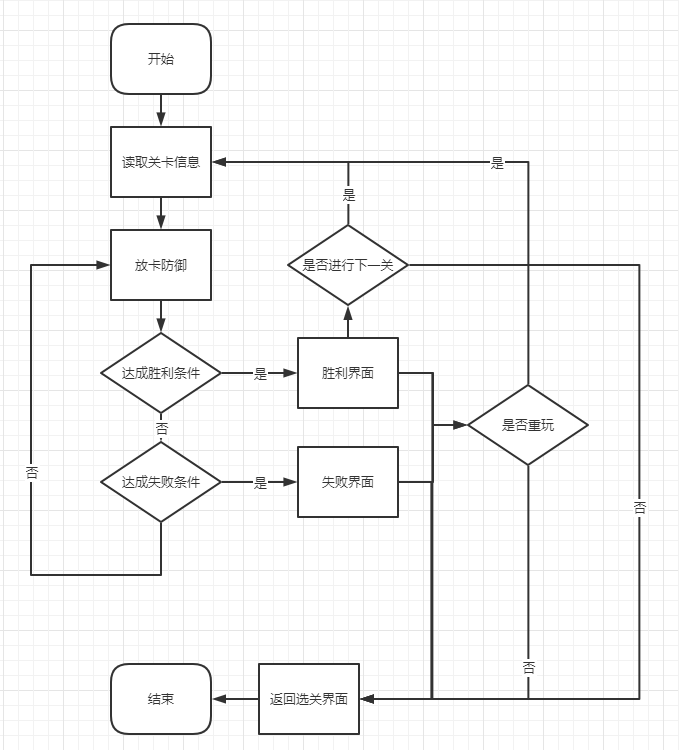
C：想做一个类似于明日方舟危机合约的系统，并且可以附加在任意关卡上。（抽象的说就是能实现在原游戏基础上直接改变一些游戏规则，比如数值增加，并且提供GUI供玩家调整）。

D：可以实现多种输入方式吗，比如除了原版的鼠标点击操作，还能同时支持键控吗。

## 3.系统总体结构

本系统作为游戏的主要系统，由各种关卡选择界面进入。

系统流程图如下



## 4.类图的设计以及说明

### 4.1 战斗场景控制器

|  |  |
| --- | --- |
| 类名 | 战斗场景控制器(GameController) |
| 整体说明 | 控制战斗场景的游戏逻辑。 |
| 成员说明 | + 当前关卡（mCurrentStage:BaseStage）  + 费用控制器(mCostController:CostController)  + 卡片建造器(mCardBuilder:CardBuilder)  + 技能控制器(mSkillController:SkillController)  + 游戏进度控制器(mProgressController:ProgressController)  + 格子表(mGridList:Grid[])  + 存活的敌方单位表(mEnemyList:BaseUnit[])  + 规则表(mRuleList:BaseRule[])  + 键位控制接口(mKeyBoardSetting: KeyBoardSetting) |
| 方法说明 | + 暂停的方法(Pause():void)  + 解除暂停的方法(Resume():void) |
| 关系说明 | 继承自MonoBehavor类 |
| 实现细节 | 一个控制类，用来顺序执行其成员的方法。  举例运用：规则表里定义了两个Rule，其中一个Rule是设定初始火苗资源为100，并且每秒额外自动回复1点火苗。第二个Rule是设定所有卡片CD减半，但消耗增加50%。这些Rule的实现均使用Rule类中的Init()与Update()方法封装起来并且被该控制器调用以实现。因此也实现了每个规则能独立执行并且与原关卡模式效果相互叠加作用，能实现自由拓展游戏规则、满足设计危机合约的需求。 |

### 4.2关卡类

#### 4.2.1 关卡基类接口

|  |  |
| --- | --- |
| 接口名 | 关卡基类接口(IBaseStage) |
| 整体说明 | 声明每个关卡必须实现的接口。 |
| 方法说明 | - 成员变量初始化方法(Init():void)  - 更新方法 (Update():void)  - 胜利条件(WinCondition():bool)  - 失败条件(LossCondition():bool) |

#### 4.2.2 关卡信息结构体

|  |  |
| --- | --- |
| 结构体名 | 关卡信息(StageInfo) |
| 整体说明 | 存储静态关卡信息数据，实现序列化，信息可转化至本地。 |
| 成员说明 | + 关卡名(mName:string)  + 轮数表(mRoundList:Round[])  + 一大波标志下标表(mWaveIndexList:int[]) |

#### 4.2.3 关卡基类

|  |  |
| --- | --- |
| 类名 | 关卡基类(BaseStage) |
| 整体说明 | 最基础的关卡类，可以作为默认关卡的模型。 |
| 成员说明 | + 关卡信息(mStageInfo: StageInfo)  + 奖励(mPrize:Prize)  - 战斗场景控制器(mGameController: GameController) |
| 方法说明 | - 成员变量初始化方法(Init():void)  - 更新方法(Update():void)  - 胜利条件(WinCondition():bool)  - 失败条件(LossCondition():bool) |
| 关系说明 | 被GameController类组合，实现关卡基类接口(IBaseStage) |
| 实现细节 | 在构造方法被调用后，在GameController的OnStart()中调用Init()方法来处理。  使用责任链模式来处理每轮出怪逻辑（根据轮数表），并且每当轮数值在一大波标志下标表有作记录时，显示一大波标志特效，当到达最后一轮时显示最后一波标志特效。  更新方法Update()可以用来实现不同关卡类的特色规则，基类里默认为空。  更新方法Update()、胜利条件WinCondition()和失败条件LossCondition()在GameController的Update()方法依次调用，一旦达成胜利条件便暂停当前战斗场景逻辑，显示出相应的页面。在达成胜利条件一刻马上调用奖励(prize)成员的Reward()方法来给予玩家对应的奖励。 |

#### 4.2.4 普通模式关卡类

|  |  |
| --- | --- |
| 类名 | 普通模式关卡类(NormalStage) |
| 整体说明 | 原版游戏的默认通关战模式。 |
| 成员说明 | - 所有敌方单位是否被清除(isClear:bool) |
| 方法说明 |  |
| 关系说明 | 继承自关卡基类(BaseStage) |
| 实现细节 | 胜利条件为所有敌方单位被清除，失败条件为敌方单位进家。  所有老鼠被清除：在Update()中先看看轮数表roundList责任链是否为空，访问控制器GameController中的是否还有存活敌方单位，若都没有，则置isClear为true，之后胜利条件部分直接判断这个布尔值即可。  敌方单位进家：遍历控制器GameController中存活的敌方单位表enemyList，任意一只的横坐标在特定范围之外即可判定进家。 |

#### 4.2.5 争分夺秒模式关卡类

|  |  |
| --- | --- |
| 类名 | 争分夺秒模式关卡类(TimeLimitStage) |
| 整体说明 | 在普通模式关卡的基础上增加一定的时间限制的模式。（为了便于维护，不继承普通模式关卡类） |
| 成员说明 | - 总时限(TotalTime:float)  - 剩余时限(TimeLeft:float)  - 是否超时(isTimeout:bool)  - 所有敌方单位是否被清除(isClear:bool) |
| 方法说明 |  |
| 关系说明 | 继承自关卡基类(BaseStage) |
| 实现细节 | 胜利条件同普通模式关卡类，失败条件在普通模式关卡类的基础上外加一条是否超时(isTimeout)  其中，剩余时限(TimeLeft:float)会在每次Update()中减去1/（正常游戏帧数），当剩余时限小于0时置isTimeout为true。 |

### 4.3 规则类

#### 4.3.1 规则基类接口

|  |  |
| --- | --- |
| 类名 | 规则基类接口(IBaseRule) |
| 整体说明 | 为战斗场景附加额外规则（如数值修改，特殊机制，只要你写得出来..） |
| 方法说明 | + 初始化方法(Init():void)  + 帧更新方法(Update():void)  + 暂停时的方法(Pause():void)  + 恢复时的方法(Resume():void)  + 销毁时的方法(Destory():void) |

#### 4.3.2 规则基类

|  |  |
| --- | --- |
| 类名 | 规则基类(BaseRule) |
| 整体说明 | 最基础的规则类，仅实现接口不完善实现细节，作为以后的规则拓展父类 |
| 成员说明 | - 战斗场景控制器(mGameController: GameController) |
| 方法说明 | + 初始化方法(Init():void)  + 帧更新方法(Update():void)  + 暂停时的方法(Pause():void)  + 恢复时的方法(Resume():void)  + 销毁时的方法(Destory():void) |
| 关系说明 | 实现规则类接口(IBaseRule)，被游戏控制类(GameController)所组合 |
| 实现细节 | 初始化方法在游戏控制类(GameController)的Awake()中被调用  帧更新方法在游戏控制类(GameController)的Update()中被调用  暂停时的方法在游戏控制类(GameController)的Pause()中被调用  恢复时的方法在游戏控制类(GameController)的Resume()中被调用  销毁时方法在游戏控制类(GameController)的Destory()中被调用  要实现细节请继承该类并重写方法 |

### 4.4 费用控制器

负责管理一个或几个游戏资源及其对应显示UI的Go，较为简单的比如PVZ、FVM的火苗系统，复杂一点的比如《亿万僵尸》的木材、石材、电力、费用等多个资源系统。

### 4.5 卡片建造器

使用建造者模式，负责管理本关卡片建造以及对应显示UI，比如卡片被选中的判断，判定一张卡片是否达到建造条件，卡片能建造时实例的生成，还有UI的动态更新（卡片剩余CD显示，费用是否够显示）。

### 4.6 技能控制器

负责管理玩家自带技能的数据、效果以及对应UI显示，如轰炸宝石的图标显示、CD动态更新、判断能否使用的逻辑、使用成功后五个爆炸海螺实例的生成等。

### 4.7 游戏进度控制器

负责管理游戏进度数值、游戏完成进度判断以及对应UI显示；如原版右下角的老鼠进度条，BOSS战时BOSS的血条显示。

### 4.8 格子

负责管理战斗场景中的游戏单元格信息，比如格子的地形状态可以为水、岩浆、空置、高地形等，格子上的物件表（如飞猪障碍，海胆障碍，蜗牛粘液，可击毁障碍），格子上的卡，格子带来的特殊属性加成等。

### 4.9 单位

在场内出现的物件应当都继承单位这个基类，比如美食、老鼠、人物、障碍等。

### 4.10 键位控制接口

在游戏外事先设置好键控映射，然后在游戏内通过按键的方式实现在鼠标悬停处直接快捷放卡等操作。例如，在外面事先把卡槽位中的1号槽绑定到A键上，然后在游戏中，鼠标悬停在某个格子上，只要按下A即可实现快捷把1号槽的卡放置到该格子上的功能。