目录

[战斗场景系统 2](#_Toc92752681)

[1.综述 2](#_Toc92752682)

[2.用户需求 2](#_Toc92752683)

[3.系统总体结构 2](#_Toc92752684)

[4.类图的设计以及说明 3](#_Toc92752685)

[4.1 战斗场景控制器 3](#_Toc92752686)

[4.2关卡类 4](#_Toc92752687)

[4.3 规则类 6](#_Toc92752688)

[4.4 费用控制器 7](#_Toc92752689)

[4.5 卡片控制器 8](#_Toc92752690)

[4.6 技能控制器 10](#_Toc92752691)

[4.7 游戏进度控制器 11](#_Toc92752692)

[4.8 格子 13](#_Toc92752693)

[4.9 单位 14](#_Toc92752694)

[4.10 键位控制监听器 15](#_Toc92752695)

[4.11 用户操作记录者 15](#_Toc92752696)

# 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。

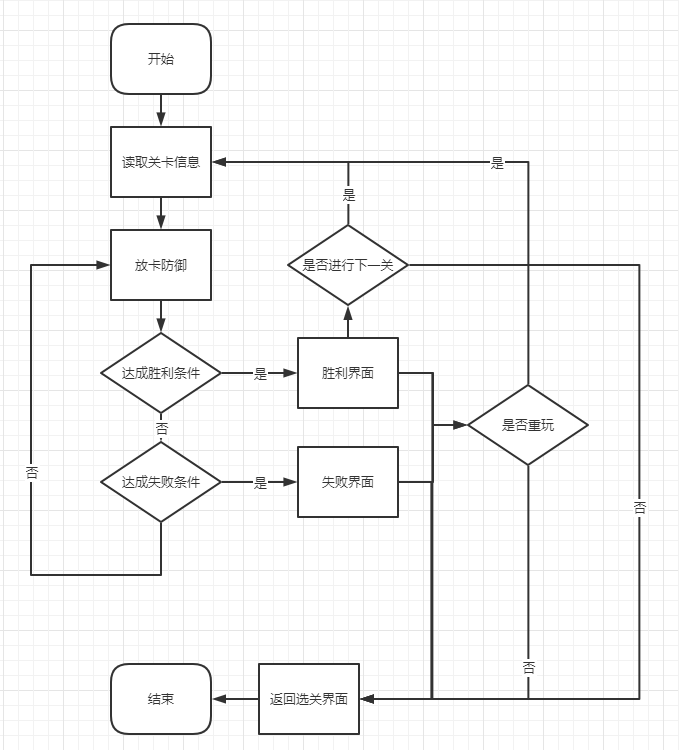
C：想做一个类似于明日方舟危机合约的系统，并且可以附加在任意关卡上。（抽象的说就是能实现在原游戏基础上直接改变一些游戏规则，比如数值增加，并且提供GUI供玩家调整）。

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

#### 4.1.1 控制器类定义

|  |  |
| --- | --- |
| 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，使用单例模式。 |
| 成员说明 | 静态成员：  + 自身单例(Instance:GameController)  动态成员：  + 当前关卡(mCurrentStage:BaseStage）  + 费用控制器(mCostController:BaseCostController)  + 卡片建造器(mCardController:BaseCardController)  + 技能控制器(mSkillController:BaseSkillController)  + 游戏进度控制器(mProgressController:BaseProgressController)  + 格子表(mGridList:BaseGrid[])  + 存活的敌方单位表(mEnemyList:BaseUnit[])  + 规则表(mRuleList:BaseRule[])  + 键位控制接口(mKeyBoardSetting:KeyBoardSetting)  + 游戏操作记录者(mRecorder:Recorder) |
| 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。 |

#### 4.1.2 控制器成员变量中自定义的管理类应继承的接口

|  |  |
| --- | --- |
| 接口名 | 控制器成员接口(IGameControllerMember) |
| 整体说明 | 控制器成员变量中自定义的管理类应继承的接口。  这些成员可能包括：当前关卡、费用控制器、卡片建造器、技能控制器、游戏进度控制器、格子等。 |
| 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) |
| 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  要实现细节请重写方法 |

### 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 |  |
| 关系说明 | 被GameController类组合，实现关卡基类接口(IBaseStage)、控制器成员接口(IGameControllerMember)。 |
| 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

|  |  |
| --- | --- |
| 接口名 | 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 |  |

#### 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 |  |
| 关系说明 | 实现规则类接口(IBaseRule)、控制器成员接口(IGameControllerMember)，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

#### 4.4.1 费用控制器接口

|  |  |
| --- | --- |
| 接口名 | 费用控制器接口(IBaseCostController) |
| 整体说明 |  |
| 方法说明 | + 获取某个费用资源方法(GetCost(string name):float)  + 修改某个费用资源方法(SetCost(string name, float val):void)  + 增加某个费用资源的方法(AddCost(string name, float val):void) |

#### 4.4.2 费用控制器基类

|  |  |
| --- | --- |
| 类名 | 费用控制器基类(BaseCostController) |
| 整体说明 | 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；根据具体需求来继承这个类处理。 |
| 成员说明 | - 战斗场景控制器(mGameController: GameController)  - 费用资源字典(mCostDict: Dictionary<string, float>)  - 火苗资源UI(mFireGo: GameObject) |
| 方法说明 |  |
| 关系说明 | 实现费用控制器接口(IBaseCostController)、控制器成员接口(IGameControllerMember)，被游戏控制类(GameController)所组合 |
| 实现细节 | Init()中先将火苗资源添加到资源字典里。  Update()中实现对UI的控制 |

### 4.5 卡片控制器

#### 4.5.1 卡片控制器接口

|  |  |
| --- | --- |
| 接口名 | 卡片控制器接口(IBaseCardController) |
| 整体说明 |  |
| 方法说明 | + 选中卡(SelectCard():void)  + 在对应的位置生成卡(Construct():void) |

#### 4.5.2 卡片控制器基类

|  |  |
| --- | --- |
| 类名 | 卡片控制器器基类(BaseCardController) |
| 整体说明 | 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）。 |
| 成员说明 | - 战斗场景控制器(mGameController: GameController)  + 所携带卡片的建造器表(mCardBuilderList: BaseCardBuilder[])  + 被选中的卡片(mSelectCard: BaseCard)  + 被选中的格子(mSelectGrid: BaseGrid) |
| 方法说明 |  |
| 关系说明 | 实现卡片控制器接口(IBaseCardController)、控制器成员接口(IGameControllerMember)，被游戏控制类(GameController)所组合。 |
| 实现细节 | Update()中应包含所携带卡片的建造器表中所有元素的Update()方法。  SelectCard()中应包含卡片本身的能否被选中的判断逻辑（4.5.4卡片基类中的CanSelect()方法），如果能则修改被选中的卡片(mSelectCard)的引用。不能则做出对应的不能提示处理。  Construct()应包含被选中卡片能否放在被选中格子（4.5.4 卡片基类中的CanConstruct()方法）的判断，如果能则应正确生成卡片Go，并且更新格子的状态信息；如果不能则要取消这次建造，置被选中格子为空，但不改变被选中卡片状态，同时做出相应的提示处理。 |

#### 4.5.3 卡片建造者基类接口

|  |  |
| --- | --- |
| 接口名 | 卡片建造者基类接口(IBaseCardBuilder) |
| 整体说明 | 游戏内管理卡片可否被选取，可否放置 |
| 方法说明 | + 能否被选取(CanSelect():bool)  + 能否被建造(CanConstruct():bool)  + 产生一个实例(Construct():void)  + 对持有实例进行初始化(InitInstance():void)  + 导出实例引用(GetResult():GameObject)  + 更新的方法(Update():void) |

#### 4.5.4 卡片建造者基类

|  |  |
| --- | --- |
| 类名 | 卡片建造者基类(BaseCardBuilder) |
| 整体说明 | 包含了游戏内建造一种卡片条件需求的信息。 |
| 成员说明 | - 战斗场景控制器(mGameController: GameController)  + 建造费用资源需求(mCostDict: Dictionary<string, float>)  - 卡片基础CD(mBaseCD: int)  + 卡片当前CD(mCD:int)  + 卡片剩余CD计数器(mCDLeft:int)  + 是否被禁用(isDisable:bool)  + 所持有的UI显示实例(mUIGo:GameObject)  + 预制体(cardPrefab:GameObject)  + 当前持有实例(currentCardGo:GameObject) |
| 方法说明 |  |
| 关系说明 | 实现卡片建造器接口(IBaseCardBuilder)、被记录的事件接口(IRecordedClass)，被卡片控制器基类(BaseCardController)所组合 |
| 实现细节 | 以上CD使用的单位是帧，基础CD从JSON中读取，并且固定下来。  当前CD默认与基础CD一样，但可以被外界影响  剩余CD计数器在小于等于0之前每帧都减一，当放置卡后令剩余CD重置为当前CD，此外，如果当游戏途中当前CD发生百分比变化，剩余CD同样要发生百分比变化。  上述CD更新方法写在Update()中，Update()方法要让卡片控制器基类(BaseCardController)调用。  CanSelect()的判断条件是：是否被禁用(isDisable==false)&&剩余CD小于等于0(mCDLeft<=0)&&建造费用需求达标（需要读取4.4.2 费用控制器实例信息并作比对）  CanConstruct()的判断条件是：这个卡是已被选中，从卡片建造器中获取&&被选中的格子对应信息满足卡片的特殊限制条件，如这个格子上没有障碍、其它卡片；水生卡片只能种在水地形格子上。  Construct()则是根据预制体产生一个实例到游戏中，并且让当前建造者持有它，以便于下一步让指挥者CardController指挥初始化这个实例。  GetResult()将当前建造者持有的实例引用导出，并且置当前建造者持有实例为空。  因此，建造一张卡的顺序应是：指挥者依次调用这个类中的Construct()、InitInstance()、GetResult()，然后完事。 |

### 4.6 技能控制器

#### 4.6.1 技能控制器基类

|  |  |
| --- | --- |
| 类名 | 技能控制器基类(BaseSkillController) |
| 整体说明 | 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。 |
| 成员说明 | - 战斗场景控制器(mGameController: GameController)  + 所携带技能的建造器表(mSkillBuilderList: BaseSkillBuilder[]) |
| 方法说明 | + 释放技能(Cast(int index):void) |
| 关系说明 | 实现控制器成员接口(IGameControllerMember)，被游戏控制类(GameController)所组合。 |
| 实现细节 | Update()中应包含所携带技能的建造器表中所有元素的Update()方法。 |

#### 4.6.2 技能建造者基类接口

|  |  |
| --- | --- |
| 接口名 | 技能建造者基类接口(IBaseSkillBuilder) |
| 整体说明 | 游戏内管理技能可否释放、释放条件。 |
| 方法说明 | + 能否被释放(CanCast():bool)  + 释放技能的具体实现(Cast():void)  + 更新的方法(Update():void) |

#### 4.6.3 技能建造者基类

|  |  |
| --- | --- |
| 类名 | 技能建造者基类(BaseSkillBuilder) |
| 整体说明 | 包含了游戏内释放一种技能的条件需求的信息。 |
| 成员说明 | - 战斗场景控制器(mGameController: GameController)  + 释放费用资源需求(mCostDict: Dictionary<string, float>)  - 技能基础CD(mBaseCD: int)  + 技能当前CD(mCD:int)  + 技能剩余CD计数器(mCDLeft:int)  + 所持有的UI显示实例(mUIGo:GameObject)  + 是否被禁用(isDisable:bool) |
| 方法说明 |  |
| 关系说明 | 实现卡片建造器接口(IBaseCardBuilder)、被记录的事件接口(IRecordedClass)，被卡片控制器基类(BaseCardController)所组合 |
| 实现细节 | 以上CD使用的单位是帧，基础CD从JSON中读取，并且固定下来。  当前CD默认与基础CD一样，但可以被外界影响  剩余CD计数器在小于等于0之前每帧都减一，当放置卡后令剩余CD重置为当前CD，此外，如果当游戏途中当前CD发生百分比变化，剩余CD同样要发生百分比变化。  上述CD更新方法写在Update()中，Update()方法要让技能控制器基类(BaseSkillController)调用。  CanCast()的判断条件是：是否被禁用(isDisable==false)&&剩余CD小于等于0(mCDLeft<=0)&&释放费用需求达标（需要读取4.4.2 费用控制器实例信息并作比对），如果有其他条件可以在子类中重写这个方法。  事实上，本节与4.5 卡片控制器的实现比较接近。 |

### 4.7 游戏进度控制器

#### 4.7.1 游戏进度控制器接口

|  |  |
| --- | --- |
| 接口名 | 游戏进度控制器接口(IBaseProgressController) |
| 整体说明 |  |
| 方法说明 | + 本次游戏进度是否完成(IsEnd():bool) |

#### 4.7.2 游戏进度控制器基类

|  |  |
| --- | --- |
| 类名 | 游戏进度控制器基类(BaseProgressController) |
| 整体说明 | 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。 |
| 成员说明 | - 战斗场景控制器(mGameController: GameController)  + 各种进度条的集合(mProgressBarList: BaseProgressBar[])  + 轮数进度条(mRoundProgressBar: RoundProgressBar) |
| 方法说明 |  |
| 关系说明 | 实现游戏进度控制器接口(IBaseProgressController)、控制器成员接口(IGameControllerMember)，被游戏控制类(GameController)所组合 |
| 实现细节 | Update()中实现对UI的控制，同时处理所持有进度条的Update()方法。  IsEnd()的具体实现默认为波数进度条与BOSS进度条的条件都结束。  注：轮数进度条与BOSS进度条也包括在各种进度条集合之中，只不过为了方便操作而添加的引用。不要重复调用Update()! |

#### 4.7.3 进度条接口

|  |  |
| --- | --- |
| 接口名 | 进度条接口(IBaseProgressBar) |
| 整体说明 |  |
| 方法说明 | + 该进度是否完成(IsFinish():bool)  + 初始化方法(Init():void)  + 更新方法(Update():void)  + 显示(Show():void)  + 隐藏(Hide():void) |

#### 4.7.4 进度条基类

|  |  |
| --- | --- |
| 类名 | 进度条基类(BaseProgressBar) |
| 整体说明 | 可以作为显示某种东西的进度，例如游戏波次显示、BOSS血量显示。要实现特化功能请继承这个类。 |
| 成员说明 | - 战斗场景控制器(mGameController: GameController) |
| 方法说明 |  |
| 关系说明 | 实现进度条接口(IBaseProgressBar)，被游戏进度基类(BaseProgressController)所组合 |
| 实现细节 |  |

#### 4.7.5 轮数进度条

|  |  |
| --- | --- |
| 类名 | 轮数进度条(RoundProgressBar) |
| 整体说明 | 具体显示游戏总波数、当前波数，以及所在轮数的信息。 |
| 成员说明 | - 波数(mWaveNum:int)  - 轮数(mRoundNum:int)  - 一大波时的轮数下标(mWaveIndexList:int[])  - 当前轮数(mCurrentRound:int) |
| 方法说明 | + 向前推进一轮(NextRound():void) |
| 关系说明 | 继承自进度条基类(BaseProgressBar) |
| 实现细节 | 在Init()中读取当前关卡波数、轮数、一大波的轮数下标信息，然后读取并实例化预制体，将波数与轮数传进去。  NextRound()方法就是直接将当前轮数(mCurrentRound)+1，如果此时轮数与一大波时的轮数下标一致则额外显示一大波的标志。这个方法应当在游戏控制器(GameController)中被调用。 |

#### 4.7.6 BOSS血条

|  |  |
| --- | --- |
| 类名 | BOSS血条(BossHpBar) |
| 整体说明 | 以BOSS血条的形式显示某个单位的血条。 |
| 成员说明 | - 目标对象(mUnit:BaseUnit)  - 目标对象消失时是否自动销毁(isAutoDestory:bool) |
| 方法说明 | + 设置为自动销毁(AutoDestoryEnable():void) |
| 关系说明 | 继承自进度条基类(BaseProgressBar) |
| 实现细节 | 在Update()中每帧判断目标对象的存活情况，如果存活则计算当前血量/最大血量百分比，不存在则置百分比为0，清空当前持有对象引用，然后判断是否需要自动销毁。  AutoDestoryEnable()方法是强制设isAutoDestory为true，主要用法是目标不存在时但因特殊原因血条不销毁，现在需要销毁之用。 |

### 4.8 格子

#### 4.8.1 格子基类

|  |  |
| --- | --- |
| 类名 | 格子基类(BaseGrid) |
| 整体说明 | 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。 |
| 成员说明 | + 主要地形状态(mMainGridState:BaseGridState)  + 其它地形状态表(mOtherGridStateList: BaseGridState[])  + 可以建造东西吗(canBuild:bool)  + 持有的卡片引用表(mCardList:BaseCard[]) |
| 方法说明 | + 切换主要地形状态(ChangeMainGridState(BaseGridState state):void)  + 新增一个地形状态(AddGridState(BaseGridState state):void)  + 移除一个地形状态(RemoveGridState(int index):void) |
| 关系说明 | 继承自MonoBehavor类 |
| 实现细节 | 主要地形状态与其它地形状态的基类是一样的，只是一个特殊的标记，比如这个地形本来就是水地形，就该把水地形状态置为主要地形状态。一些特殊的实现如水转陆也可以通过改变主要地形状态。换言之，主要地形状态使用的也是状态模式。  其它地形状态更像是一类临时的BUFF类效果，比如在地面上持续的燃烧效果。尽管它也使用地形状态基类但它不是状态模式。  所有地形状态都有更新canBuild的权限，canBuild的实现逻辑应为：每帧默认置为true，然后由各种地形状态的Update()来决定要不要置为false。此外，canBuild状态默认与当前格子上有没有卡片是无关的，它只是代表这个格子在没有卡片时能不能放东西的状态。  RemoveGridState方法给地形状态基类调用。 |

#### 4.8.2 地形状态基类

|  |  |
| --- | --- |
| 类名 | 地形状态基类(BaseGridState) |
| 整体说明 | 用来处理不同状态下单位的动画播放及其他逻辑 |
| 成员说明 | - 持有该状态的格子(mGrid:BaseGrid)  + 自身所在格子下标(index:int) |
| 方法说明 | + 实现状态(Update():void) |
| 关系说明 | 被格子基类(BaseGrid)所组合。 |
| 实现细节 | 具体动作实现请继承这个类并重写这个方法 |

### 4.9 单位

#### 4.9.1 单位基类

|  |  |
| --- | --- |
| 类名 | 单位基类(BaseUnit) |
| 整体说明 | 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。 |
| 成员说明 | + 基础生命值(mBaseHp:float)  + 最大生命值(mMaxHp:float)  + 当前生命值(mCurrentHp:float)  + 基础攻击力(mBaseAttack:float)  + 当前攻击力(mCurrentAttack:float)  + 基础攻击速度(mBaseAttackSpeed:float)  + 当前攻击速度(mCurrentAttackSpeed:float)  + 当前攻击间隔(mAttackCD:int)  + 攻击间隔计数器(mAttackCDLeft:int)  + 高度(mHeight:int)  + 当前动作状态(mCurrentActionState:BaseActionState)  + |
| 方法说明 | + 切换动作状态(SetActionState(BaseActionState state):void) |
| 关系说明 | 继承自MonoBehavor类 |
| 实现细节 | 攻击速度的值为每秒攻击的次数，因此当前攻击间隔=int(60/(当前攻击速度))。  使用状态模式来处理单位的不同动作，如待机、移动、攻击。 |

#### 4.9.2 动作状态基类

|  |  |
| --- | --- |
| 类名 | 动作状态基类(BaseActionState) |
| 整体说明 | 用来处理不同状态下单位的动画播放及其他逻辑 |
| 成员说明 |  |
| 方法说明 | + 实现动作状态(DoAction():void)  + 仅仅只是播放动画(PlayAnimation():void) |
| 关系说明 | 被单位基类(BaseUnit)所组合。 |
| 实现细节 | 具体动作实现请继承这个类并重写这个方法 |

### 4.10 键位控制监听器

|  |  |
| --- | --- |
| 类名 | 键位控制监听器(KeyListener) |
| 整体说明 | 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。 |
| 成员说明 | - 战斗场景控制器(mGameController:GameController)  + 每个键位的监听(mActionListenerList:ActionListener[])  + 是否设定为快捷操作(isShortcut:bool) |
| 方法说明 | - 按下某个键位所执行的方法(execute(int key): void)  - 初始化的方法(init():void)  + 每帧更新的方法(update():void) |
| 关系说明 | 被战斗场景控制器(GameController)所组合。 |
| 实现细节 | 数组mActionListenerList大小为128，其下标对应的是被激活的键位的ASCII值。 |

### 4.11 用户操作记录者

#### 4.11.1 被记录的事件接口

|  |  |
| --- | --- |
| 接口名 | 被记录的事件接口(IRecordedClass) |
| 整体说明 | 来自玩家任何形式的输入最终会被转化为调用以下六个方法，比如鼠标点击选卡这个操作会被转化为调用对应卡片建造者的焦点按下操作，继承这个接口的类在调用下述六个方法时会被记录下来。 |
| 方法说明 | + 焦点按下(OnFocusDown():void) + 焦点松开(OnFocusUp():void) + 焦点位于按下与松开的期间(OnFocusDrag():void) + 焦点移入(OnFocusEnter():void) + 焦点悬停(OnFocusOver():void) + 焦点移出(OnFoucsExit():void) |

#### 4.11.2 用户操作记录者

|  |  |
| --- | --- |
| 类名 | 用户操作记录者(Recorder) |
| 整体说明 | 正常游戏模式下，用来获取用户的操作并记录关键操作、管理并保存。  在回放模式下，通过读取已保存的操作，来实现完美复刻玩家操作。 |
| 成员说明 | + 每帧信息表(mRecordInfoList: RecordInfo[]) + 是否为回放模式(isReplay:bool) + 当前持有帧下标(index:int) + 所有会被记录的类的表(mRecordedClassList:IRecordedClass[]) |
| 方法说明 | + 获取当前帧信息(GetCurrentFrameRecordInfoList(): RecordInfo[]) + 在本帧内添加记录信息(AddRecordInfoToCurrentFrame(RecordInfo info):void) |
| 关系说明 | 被战斗场景控制器(GameController)所组合。 |
| 实现细节 |  |

#### 4.11.3 记录信息结构体

|  |  |
| --- | --- |
| 结构体名 | 记录信息(RecordInfo) |
| 整体说明 | 存储某一帧中记录的特定操作信息数据，实现序列化，信息可转化至本地。 |
| 成员说明 | + 帧数(mNum:int) + 被操作类在被记录的类的表的下标(index:int) + 操作类型(mOperationType:OperationType) |

#### 4.11.4 操作方法枚举

|  |  |
| --- | --- |
| 枚举名 | 操作方法类型(OperationType:int) |
| 整体说明 | 以枚举的方式对应于被记录事件接口的六个方法。 |
| 成员说明 | + 焦点按下(OnFocusDown=0) + 焦点松开(OnFocusUp=1) + 焦点位于按下与松开的期间(OnFocusDrag=2) + 焦点移入(OnFocusEnter=3) + 焦点悬停(OnFocusOver=4) + 焦点移出(OnFoucsExit=5) |