

|  |
| --- |
| MineWar |
| 设计书 |
| 软件开发系列文档 |
| Ver 1.0.0  zhaoyg  2016/9/16 |

|  |
| --- |
| All rights reserved, 2016 |

变更履历

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 序号 | 版本 | 变更内容 | 变更人 | 日期 |
| 1 | 0.5.0 | 创建文档，定义基本章节 | zhaoyg | 2016-8-18 |
| 2 | 0.7.0 | 追加系统架构、应用框架设计 | XA | 2016-8-18 |
| 3 | 0.7.1 | 追加UI库设计、ViewStack | XC | 2016-8-22 |
| 4 | 0.7.2 | 追加Graphic库设计 | XD | 2016-8-22 |
| 5 | 0.7.3 | 追加Log设计 | XA | 2016-8-22 |
| 6 | 0.8.0 | 追加Game设计 | XB | 2016-8-23 |
| 7 | 0.8.1 | 追加UI设计 | XE | 2016-8-23 |
| 8 | 0.9.0 | 评审后修改 | ALL | 2016-9-7 |
| 9 | 1.0.0 | 更新设计，与最新实现保持一致 | ALL | 2016-9-16 |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

目录

[目录 3](#_Toc533248404)

[1 概述 8](#_Toc533248405)

[2 开发环境构成 8](#_Toc533248406)

[3 系统架构 9](#_Toc533248407)

[3.1 App Framework 10](#_Toc533248408)

[3.2 Disp Driver 10](#_Toc533248409)

[3.3 Graphic Lib 10](#_Toc533248410)

[3.4 UI Lib 10](#_Toc533248411)

[3.5 UI Views 10](#_Toc533248412)

[3.6 Game Core 10](#_Toc533248413)

[4 App框架 10](#_Toc533248414)

[4.1 功能 10](#_Toc533248415)

[4.2 主要处理流程 10](#_Toc533248416)

[4.3 时序 12](#_Toc533248417)

[4.3.1 启动 12](#_Toc533248418)

[4.3.2 触摸操作 12](#_Toc533248419)

[4.3.3 描画 13](#_Toc533248420)

[4.4 API 14](#_Toc533248421)

[4.4.1 App\_startup 14](#_Toc533248422)

[4.4.2 App\_exit 14](#_Toc533248423)

[4.4.3 App\_touchesBegan 15](#_Toc533248424)

[4.4.4 App\_touchesEnded 15](#_Toc533248425)

[4.4.5 App\_touchesMoved 15](#_Toc533248426)

[4.4.6 App\_draw 15](#_Toc533248427)

[4.4.7 App\_sendEvent 16](#_Toc533248428)

[4.4.8 App\_isCalEnable 16](#_Toc533248429)

[5 UI库 16](#_Toc533248430)

[5.1 C Class 16](#_Toc533248431)

[5.1.1 封装 17](#_Toc533248432)

[5.1.2 继承 17](#_Toc533248433)

[5.1.3 多态 18](#_Toc533248434)

[5.1.4 构造函数 18](#_Toc533248435)

[5.1.5 静态成员和静态函数 18](#_Toc533248436)

[5.1.6 类方法 19](#_Toc533248437)

[5.2 UI Architecture 19](#_Toc533248438)

[5.2.1 Widget类 20](#_Toc533248439)

[5.2.2 Button类 21](#_Toc533248440)

[5.2.3 TouchMtx 22](#_Toc533248441)

[5.2.4 View类 22](#_Toc533248442)

[5.2.5 VXxx类 24](#_Toc533248443)

[5.3 控件触发event 25](#_Toc533248444)

[5.3.1 View 26](#_Toc533248445)

[5.3.2 Button 26](#_Toc533248446)

[5.3.3 TouchMtx 27](#_Toc533248447)

[6 ViewStack 27](#_Toc533248448)

[6.1 迁移 27](#_Toc533248449)

[6.1.1 CALL-RETURN 27](#_Toc533248450)

[6.1.2 CHANGE 27](#_Toc533248451)

[6.2 回调 28](#_Toc533248452)

[6.3 注册 28](#_Toc533248453)

[6.4 API 28](#_Toc533248454)

[6.4.1 regView 28](#_Toc533248455)

[6.4.2 setInPara 29](#_Toc533248456)

[6.4.3 push1stView 29](#_Toc533248457)

[6.4.4 callView 29](#_Toc533248458)

[6.4.5 returnView 29](#_Toc533248459)

[6.4.6 changeView 30](#_Toc533248460)

[6.4.7 getTopView 30](#_Toc533248461)

[6.4.8 transfer 30](#_Toc533248462)

[7 Graphic库 30](#_Toc533248463)

[7.1 Resource 30](#_Toc533248464)

[7.2 Graphic API 31](#_Toc533248465)

[7.2.1 drawImage 31](#_Toc533248466)

[7.2.2 drawImage2 31](#_Toc533248467)

[7.2.3 drawText 32](#_Toc533248468)

[7.2.4 drawText2 32](#_Toc533248469)

[7.2.5 drawTextC 32](#_Toc533248470)

[7.2.6 drawTextR 32](#_Toc533248471)

[7.2.7 getTextWidth 33](#_Toc533248472)

[7.2.8 drawRectFill 33](#_Toc533248473)

[7.2.9 drawRectFill2 33](#_Toc533248474)

[7.2.10 drawRect 34](#_Toc533248475)

[7.2.11 allocResArray 34](#_Toc533248476)

[7.2.12 freeResArray 34](#_Toc533248477)

[7.2.13 addImage 34](#_Toc533248478)

[7.2.14 regVram 35](#_Toc533248479)

[7.3 Color API 35](#_Toc533248480)

[7.3.1 RGBA 35](#_Toc533248481)

[7.3.2 RGB 35](#_Toc533248482)

[7.3.3 BGR 36](#_Toc533248483)

[7.3.4 setBGR 36](#_Toc533248484)

[8 Game Core模块 37](#_Toc533248485)

[8.1 模块构成 37](#_Toc533248486)

[8.1.1 启动/终止API 37](#_Toc533248487)

[8.1.2 界面操作API 38](#_Toc533248488)

[8.1.3 数据取得API 41](#_Toc533248489)

[8.1.4 周期处理 47](#_Toc533248490)

[8.2 双向链表 48](#_Toc533248491)

[8.2.1 数据结构 48](#_Toc533248492)

[8.2.2 通用算法 49](#_Toc533248493)

[8.2.3 API 51](#_Toc533248494)

[8.3 数据结构 53](#_Toc533248495)

[8.3.1 地图 53](#_Toc533248496)

[8.3.2 队列 54](#_Toc533248497)

[8.3.3 敌人 54](#_Toc533248498)

[8.3.4 地雷 55](#_Toc533248499)

[8.3.5 庄稼 55](#_Toc533248500)

[8.3.6 房屋 56](#_Toc533248501)

[8.3.7 子弹 57](#_Toc533248502)

[8.3.8 地雷炮 58](#_Toc533248503)

[8.4 算法 58](#_Toc533248504)

[8.4.1 敌人队列移动 58](#_Toc533248505)

[8.4.2 游戏难度 61](#_Toc533248506)

[8.5 配置 61](#_Toc533248507)

[8.5.1 基本数据 62](#_Toc533248508)

[8.5.2 地图 63](#_Toc533248509)

[8.5.3 敌人队列 63](#_Toc533248510)

[9 UI实现 63](#_Toc533248511)

[9.1 App界面迁移 63](#_Toc533248512)

[9.2 游戏界面操作及状态迁移 63](#_Toc533248513)

[9.2.1 状态迁移 63](#_Toc533248514)

[9.2.2 工具栏 65](#_Toc533248515)

[9.2.3 操作对象及菜单项 65](#_Toc533248516)

[9.3 游戏状态迁移 66](#_Toc533248517)

[10 调试Log 67](#_Toc533248518)

[10.1 类型 67](#_Toc533248519)

[10.2 实现 67](#_Toc533248520)

[10.3 API 67](#_Toc533248521)

图表目录

[图 1‑1 MineWar游戏 8](#_Toc533248522)

[图 2‑1 开发环境 9](#_Toc533248523)

[图 3‑1 系统Block图 9](#_Toc533248524)

[图 4‑1 APP主流程 11](#_Toc533248525)

[图 4‑2 启动时序 12](#_Toc533248526)

[图 4‑3 触摸操作处理时序 13](#_Toc533248527)

[图 4‑4 描画时序 14](#_Toc533248528)

[图 5‑1 C Class 17](#_Toc533248529)

[图 5‑2 UI Lib Architecture 19](#_Toc533248530)

[图 5‑3 Widget类 20](#_Toc533248531)

[图 5‑4 Button类 21](#_Toc533248532)

[图 5‑5 TouchMtx Class 22](#_Toc533248533)

[图 5‑6 View类 23](#_Toc533248534)

[图 5‑7 VXxx类 25](#_Toc533248535)

[图 5‑8 控件状态迁移及触发event检测 26](#_Toc533248536)

[图 6‑1 CALL-RETURN迁移 27](#_Toc533248537)

[图 6‑2 CHANGE迁移 28](#_Toc533248538)

[图 7‑1 图片资源存储 31](#_Toc533248539)

[图 8‑1 Game Core模块框图 37](#_Toc533248540)

[图 8‑2 Game周期处理 48](#_Toc533248541)

[图 8‑3 双向链表 49](#_Toc533248542)

[图 8‑4 查找 49](#_Toc533248543)

[图 8‑5 查找算法 50](#_Toc533248544)

[图 8‑6 庄稼状态迁移 56](#_Toc533248545)

[图 8‑7 子弹移动（非垂直） 57](#_Toc533248546)

[图 8‑8 子弹移动（垂直） 58](#_Toc533248547)

[图 8‑9 地图-移动方向转换算法 60](#_Toc533248548)

[图 8‑10 游戏难度因素 61](#_Toc533248549)

[图 8‑11 配置文件 62](#_Toc533248550)

[图 9‑1 应用界面迁移 63](#_Toc533248551)

[图 9‑2 游戏界面操作 64](#_Toc533248552)

[图 9‑3 游戏界面操作状态迁移 64](#_Toc533248553)

[图 9‑4 游戏状态迁移 66](#_Toc533248554)

[表 5‑1 Widget类成员变量 20](#_Toc533248555)

[表 5‑2 Widget类成员函数 21](#_Toc533248556)

[表 5‑3 Button类成员变量 21](#_Toc533248557)

[表 5‑4 Button类成员函数 22](#_Toc533248558)

[表 5‑5 TouchMtx类成员变量 22](#_Toc533248559)

[表 5‑6 TouchMtx类成员函数 22](#_Toc533248560)

[表 5‑7 View类成员变量 23](#_Toc533248561)

[表 5‑8 View类成员函数 24](#_Toc533248562)

[表 5‑9 VXxx类成员变量 25](#_Toc533248563)

[表 5‑10 VXxx类成员函数 25](#_Toc533248564)

[表 8‑1 地图元素 53](#_Toc533248565)

[表 8‑2 队列元素 54](#_Toc533248566)

[表 8‑3 敌人信息 54](#_Toc533248567)

[表 8‑4 地雷信息 55](#_Toc533248568)

[表 8‑5庄稼信息 55](#_Toc533248569)

[表 8‑6 房屋信息 56](#_Toc533248570)

[表 8‑7 子弹信息 57](#_Toc533248571)

[表 8‑8 地雷炮信息 58](#_Toc533248572)

[表 8‑9 基本配置数据 62](#_Toc533248573)

[表 9‑1 操作状态定义 65](#_Toc533248574)

[表 9‑2 操作对象及菜单项 65](#_Toc533248575)

[表 10‑1 Log类型 67](#_Toc533248576)

# 概述

本文档为MineWar应用相关设计，包括开发环境、应用框架、UI库、描画库、Log功能、游戏核心数据结构和算法、主要流程和时序等的设计。



图 ‑1 MineWar游戏

# 开发环境构成

MineWar应用为Win32 Console应用，使用Visual Studio 2008开发。

* 开发环境由模拟面板（Simulator Panel）应用和VS2008开发的Console应用组成；
* 模拟面板应用和MineWar应用采用共享内存进行通信。通信内容为触摸操作触发以及显示数据（RGB888格式）；
* 面板通信使用Disp Driver DLL提供的API实现。



图 ‑1 开发环境

# 系统架构

MineWar应用包括App Framework、Disp Driver、Graphic Lib、UI Lib、UI View、Game Core几个部分。

各模块划分以及调用关系参见下图。



图 ‑1 系统Block图

## App Framework

实现App顶层处理，包括启动（包括文件、资源的加载）、Touch事件处理、显示数据刷新等处理。

## Disp Driver

实现和Simulator Panel的通信，包括Touch操作检测、显示数据发送。该部分属于Simulator Panel的一部分，已经开发完成，因此不在本次开发范围内。

## Graphic Lib

实现图片、文本、矩形的描画API，包括图片资源的加载和管理。

## UI Lib

实现按钮、Touch矩阵、视图等控件的抽象以及画画迁移管理。

## UI Views

使用UI Lib和Graphic Lib，实现各个具体画面（例如开机画面、设定画面、游戏画面等），通常包括操作和描画两部分。

## Game Core

实现游戏核心数据结构以及算法。

# App框架

## 功能

App框架实现应用顶层框架，包括启动（例如开机画面、游戏画面等）、周期处理、Touch触发处理、描画以及显示数据发送等。

## 主要处理流程

* 为了简单起见，App采用单线程。完成App启动处理后，进入主循环；
* 在主循环中主要完成Touch操作读取及处理、周期性处理、描画并发送数据；
* 循环基本周期为10ms，描画周期为50ms。由于描画消耗较多的时间，因此在描画周期内未执行sleep处理。



图 ‑1 APP主流程

## 时序

### 启动

启动时序如下：

1. 初始化Graphic Lib，加载Image资源；
2. 加载地图、队列以及标定数据文件；
3. 创建View Stack并初始化；
4. 创建各个画面View并注册到View Stack。



图 ‑2 启动时序

### 触摸操作

触摸操作时序如下：

1. 取得Touch状态以及坐标；
2. 如果当前状态不等于前次状态且当前状态为On，则进行Touch On处理；
3. App Framework调用View Stack中最上View的onTouchOn处理；
4. View控件onTouchOn处理遍历控件表，查找到坐标所在控件；
5. 调用Touch Matrix的onTouchOn处理，检测出EV\_MTX\_DOWN触发；
6. 调用Game View的procEvent来处理触发。例如调用game\_add\_landmine；
7. Touch On处理完成。

Touch Off/Touch Move处理类似，本处不做详细说明。



图 ‑3 触摸操作处理时序

### 描画

描画时序如下：

1. main函数中调用APP\_draw()函数；
2. App Framework调用View Stack中最上层View（例如Game View）的onDraw函数；
3. Game View调用Graphic Lib的API进行描画；
4. 描画所需的数据调用Game Core的get函数取得（可能需要很多信息，本处不详细描述）；
5. 完成描画后，调用Disp Driver的数据发送函数发送屏幕数据到Simulator模拟面板应用；
6. 完成一次描画。



图 ‑4 描画时序

## API

App Framework实现的API如下：

### App\_startup

|  |  |  |
| --- | --- | --- |
| API名称 | App\_startup | |
| 原型声明 | void App\_startup(void) | |
| 功能描述 | APP启动，完成各模块初始化 | |
| 返回值 | - | - |
| 注意事项 | 必须在程序中最先调用 | |

### App\_exit

|  |  |  |
| --- | --- | --- |
| API名称 | App\_exit | |
| 原型声明 | void App\_exit(void) | |
| 功能描述 | APP退出，完成各种动态分配资源的释放 | |
| 返回值 | - | - |
| 注意事项 | 无 | |

### App\_touchesBegan

|  |  |  |
| --- | --- | --- |
| API名称 | App\_touchesBegan | |
| 原型声明 | void App\_touchesBegan(int x, int y) | |
| 功能描述 | 触摸按下处理 | |
| 参数(in) | x | x坐标，0～SCR\_WIDTH-1 |
| 参数(in) | y | y坐标，0～SCR\_HEIGHT-1 |
| 返回值 | - | - |
| 注意事项 | 左上角为坐标原点 | |

### App\_touchesEnded

|  |  |  |
| --- | --- | --- |
| API名称 | App\_touchesEnded | |
| 原型声明 | void App\_touchesEnded(int x, int y) | |
| 功能描述 | 触摸释放处理 | |
| 参数(in) | x | x坐标，0～SCR\_WIDTH-1 |
| 参数(in) | y | y坐标，0～SCR\_HEIGHT-1 |
| 返回值 | - | - |
| 注意事项 | 左上角为坐标原点 | |

### App\_touchesMoved

|  |  |  |
| --- | --- | --- |
| API名称 | App\_touchesMoved | |
| 原型声明 | void App\_touchesMoved(int x, int y) | |
| 功能描述 | 触摸滑动处理 | |
| 参数(in) | x | x坐标，0～SCR\_WIDTH-1 |
| 参数(in) | y | y坐标，0～SCR\_HEIGHT-1 |
| 返回值 | - | - |
| 注意事项 | 左上角为坐标原点 | |

### App\_draw

|  |  |  |
| --- | --- | --- |
| API名称 | App\_draw | |
| 原型声明 | void App\_draw(void) | |
| 功能描述 | APP描画处理（通常是描画ViewStack中的顶层画面） | |
| 返回值 | - | - |
| 注意事项 | 无 | |

### App\_sendEvent

|  |  |  |
| --- | --- | --- |
| API名称 | App\_sendEvent | |
| 原型声明 | void App\_sendEvent(int event) | |
| 功能描述 | 向ViewStack发送事件 | |
| 参数(in) | event | 事件，目前定义的事件如下：  EV\_TIME\_TICK：10ms周期事件  EV\_PAUSE：应用暂停（进入后台）  EV\_RESUME：应用恢复（进入前台） |
| 返回值 | - | - |
| 注意事项 | 无 | |

### App\_isCalEnable

|  |  |  |
| --- | --- | --- |
| API名称 | App\_isCalEnable | |
| 原型声明 | bool App\_isCalEnable(void) | |
| 功能描述 | 判断Calibration是否有效 | |
| 返回值 | bool | true：Calibration模式/false：用户模式 |
| 注意事项 | 无 | |

# UI库

UI Lib实现控件（Widget）、视图（View），采用面向对象方式实现。

## C Class

虽然我们采用面向对象方式实现，但我们并没有打算使用C++来实现UI Lib。我们只是希望以类似的形式来实现，我们并不期望实现完美的封装、继承、多态。



图 ‑1 C Class

### 封装

* 采用C语言的结构体来实现类，类成员变量采用结构体成员变量来实现。类成员函数采用函数指针来实现；
* 成员函数如何来访问对象能？在C语言中没有this指针这样的实现。所以，我们只能通过参数将对象地址传递给成员函数。成员函数的第一个参数为对象指针。例如：

static void onTouchOn(Widget\_t \*\_self, int x0, int y0)；

关于private、public、protect等访问控制，我们就不实现了。

### 继承

* 子类的第一个成员变量是父类对象，这样就继承了父类的成员变量和方法；
* 使用父类指针指向子类对象时，也可以访问子类对象。

typedef struct{

......

} parent\_t;

typedef struct{

parent\_t super; /\***父类对象**\*/

......

} child\_t;

只能实现单继承，无法实现多继承。

另外，关于继承时父类的成员变量直接引用，限于C语言的限制，我们无法实现。

### 多态

* 使用父类指针来访问子类方法。
* 子类实现新的成员函数，之后将父类成员函数指针指向子类定义的函数。这样，即使使用父类指针，也可以调用子类的成员函数了。

但由此产生一个问题，父类指针被替换后，再无法调用父类的成员函数了。很多情况下，在子类重写的函数中还是需要调用父类的成员函数的。因此我们必须保留父类的成员函数指针，因此我们在子类中增加一个父类对象，专门用于保存父类成员函数指针。但这样也产生了一个问题：增加的对象的成员变量浪费了一定的存储空间。还好我们对存储空间没有限制，虽然浪费了一些空间，但换来了较为简单的实现。

typedef struct{

......

} parent\_t;

typedef struct{

parent\_t super;

parent\_t super\_f; /\***仅用来存储父类的成员函数**\*/

......

} child\_t;

### 构造函数

* 使用类名\_init(\*\_self)来作为构造函数，例如：

void Widget\_init(Widget\_t \*\_self);

在构造函数中成员函数指针才指向相应的函数，所以，在使用对象方法之前，必须显示调用构造函数。

### 静态成员和静态函数

* 静态成员不定义在类（也就是struct）中，定义在类实现的源文件中；
* 静态函数也采用函数指针实现，但第一参数不再是对象指针。所以静态函数无法访问对象成员变量。

typedef struct{

......

void (\*setMoveThreshold)(int); /\***静态函数**\*/

} View\_t;

static int mMoveThreshold = 35; /\***静态成员**\*/

static void setMoveThreshold(int threshold)

{

mMoveThreshold = threshold;

}

### 类方法

我们无法像C++那样使用类名来直接调用类方法。为了模仿的稍微像一些，我们定义一个名称和类名（struct名）类似的全局变量，之后使用这个全局变量来调用类方法（静态函数）。

View\_t View;

View\_init(&View); //通常在APP启动时初始化一次即可

View.setMoveThreshold(32); /\***调用类方法**\*/

## UI Architecture

UI Lib包括Widget（控件）、视图（View）、视图栈（ViewStack）的实现。各个类的继承和依赖关系如下：



图 ‑2 UI Lib Architecture

* Widget是控件父类，也是视图的父类；
* Button类和TouchMtx类继承自Widget；
* ViewStack类需要使用（依赖）View类；
* 各个具体的画面类（UI Views）继承自View类实现。

### Widget类

Widget类是控件的基本抽象，是所有操作类控件以及视图的基类。



图 ‑3 Widget类

表 ‑1 Widget类成员变量

|  |  |  |
| --- | --- | --- |
| 成员变量名称 | 类型 | 描述 |
| mId | int | 控件ID |
| mX | int | X坐标 |
| mY | int | Y坐标 |
| mW | int | 宽度 |
| mH | int | 高度 |
| mTX | int | 当前触摸点x坐标 |
| mTY | int | 当前触摸点y坐标 |
| mState | int | 状态，包括TOUCH\_ON/ TOUCH\_OFF |
| mProperty | int | 属性，包括DISABLE和HIDE位 |
| mMsg | Msg\_t \* | 消息指针，用于触发event暂存 |

表 ‑2 Widget类成员函数

|  |  |
| --- | --- |
| 成员函数 | 描述 |
| isInRect | 判断坐标是否在控件范围内 |
| show | 取消隐藏属性 |
| hide | 设置隐藏属性 |
| isHiden | 判断是否隐藏 |
| enable | 控件设置为有效 |
| disable | 控件设置为无效 |
| isDisabled | 判断是否有效 |
| onTouchOn | Touch On处理 |
| onTouchOff | Touch Off处理 |
| onTouchMove | Touch Move处理 |
| onTouchClear | Touch状态清除为TOUCH\_OFF |
| setEvent | 设置控件触发event |
| getEvent | 取得控件触发event |
| clearEvent | 清除控件触发event |
| getId | 取得控件ID |

### Button类

Button类为按钮类操作控件抽象，用于实现点击操作。



图 ‑4 Button类

表 ‑3 Button类成员变量

|  |  |  |
| --- | --- | --- |
| 成员变量名称 | 类型 | 描述 |
| super | Widget\_t | 父类对象 |
| super\_f | Widget\_t | 父类对象（用于备份父类成员函数） |

表 ‑4 Button类成员函数

|  |  |
| --- | --- |
| 成员函数 | 描述 |
| onTouchOn | Touch On处理 |
| onTouchOff | Touch Off处理 |

### TouchMtx

TouchMtx类为矩阵操作控件抽象，用于实现列表、地图操作。



图 ‑5 TouchMtx Class

表 ‑5 TouchMtx类成员变量

|  |  |  |
| --- | --- | --- |
| 成员变量名称 | 类型 | 描述 |
| super | Widget\_t | 父类对象 |
| super\_f | Widget\_t | 父类对象（用于备份父类成员函数） |
| gw | double | Matrix格宽度 |
| gh | double | Matrix格高度 |
| gx | int | 格x坐标，gx=(tx-x)/gw |
| gy | int | 格y坐标，gy=(ty-y)/gh |

表 ‑6 TouchMtx类成员函数

|  |  |
| --- | --- |
| 成员函数 | 描述 |
| onTouchOn | Touch On处理 |
| onTouchOff | Touch Off处理 |

### View类

View类即视图类，是所有操作画面的基本抽象。



图 ‑6 View类

表 ‑7 View类成员变量

|  |  |  |
| --- | --- | --- |
| 成员变量名称 | 类型 | 描述 |
| super | Widget\_t | 父类对象 |
| super\_f | Widget\_t | 父类对象（用于备份父类成员函数） |
| mTX0 | int | 当前触摸点x坐标 |
| mTY0 | int | 当前触摸点y坐标 |
| mRefreshFlag | bool | 界面刷新标志 |
| mMoveDirection | int | 滑动方向 |
| mWidgetArray | Widget\_t\*\* | 控件数组 |
| mWidgetCount | int | 控件总数 |
| mTransfer | Transfer\_t | View迁移请求 |
| mInputPara | Transfer\_t | View迁移参数 |

表 ‑8 View类成员函数

|  |  |
| --- | --- |
| 成员函数 | 描述 |
| onCreate | 创建，在View入栈时调用 |
| onEnter | 进入，在View入栈时调用 |
| onExit | 退出，在View出栈时调用 |
| onShow | 显示，在画面显示前调用 |
| onHide | 隐藏，在画面被覆盖前调用 |
| onTouchOn | Touch On处理 |
| onTouchOff | Touch Off处理 |
| onTouchMove | 滑动处理 |
| onDraw | 描画函数 |
| procEvent | 控件触发消息处理回调 |
| sendEvent | 通常消息处理回调 |
| regWidgets | 注册控件 |
| setInputPara | 设置迁移参数 |
| setTransfer | 设置迁移请求 |
| getTransfer | 取得迁移请求 |
| clearTransfer | 清除迁移请求 |
| getRefreshFlag | 取得刷新标志 |
| clearRefreshFlag | 清除刷新标志 |
| setMoveDirection | 设置滑动检测方向 |
| setMoveThreshold | 设置滑动检测阈值 |

### VXxx类

VXxx为具体的画面实现类，例如开机画面、游戏画面、帮助画面等。具体画面类都派生自View类。

* 具体画面实现可以根据需要增加成员变量和成员函数；
* onCreate、onEnter、onDraw、procEvent、sendEvent几个函数必须进行覆盖（override），其余函数可以根据需要进行覆盖。



图 ‑7 VXxx类

表 ‑9 VXxx类成员变量

|  |  |  |
| --- | --- | --- |
| 成员变量名称 | 类型 | 描述 |
| super | View\_t | 父类对象 |
| super\_f | View\_t | 父类对象（用于备份父类成员函数） |
| mXXX | X\_t | 具体的画面可以根据需要添加必要的成员变量 |

表 ‑10 VXxx类成员函数

|  |  |
| --- | --- |
| 成员函数 | 描述 |
| onCreate | 创建，在View入栈时调用 |
| onEnter | 显示，在画面显示前调用 |
| onDraw | 描画函数 |
| procEvent | 控件触发消息处理回调 |
| sendEvent | 通常消息处理回调 |

## 控件触发event

控件触发event分为触摸和滑动两种。检测规则如下：

1. 初始状态为TOUCH\_OFF；
2. 用户触摸屏幕时，检测出EV\_XXX\_DOWN触发；
3. 如果用户没有滑动或滑动距离较小，之后释放屏幕，则检测出EV\_XXX\_UP触发；
4. 如果用户触摸后滑动距离超过指定阈值，则检出View滑动触发EV\_SLIDE\_X；
5. 为了避免一次操作检测出两个触发，当View检测到滑动后，控件状态将被Clear为TOUCH\_OFF状态，但不会发送EV\_XXX\_UP触发，例如EV\_KEY\_UP或EV\_MTX\_UP。



图 ‑8 控件状态迁移及触发event检测

### View

* 触发：

EV\_SLIDE\_L：向左滑动

EV\_SLIDE\_R：向右滑动

EV\_SLIDE\_U：向上滑动

EV\_SLIDE\_D：向下滑动

* 检测方向设置：

View仅支持水平或垂直方向的滑动检测，需要调用setMoveDirection来指定滑动检测方向。

SLIDE\_DET\_NONE：不检测滑动

SLIDE\_DET\_LR：检测左右滑动

SLIDE\_DET\_UD：检测上下滑动

* 滑动检测阈值

检测出滑动触发的移动量使用类方法setMoveThreshold进行设置。

### Button

* EV\_KEY\_UP：按钮释放
* EV\_KEY\_DOWN：按钮按下

### TouchMtx

* EV\_MTX\_UP：Matrix释放，带有(x, y)参数
* EV\_MTX\_DOWN：Matrix按下，带有(x, y)参数

坐标(x, y)的原点在左上角，向右和向下为x和y的正方向。

# ViewStack

ViewStack类实现画面迁移，包括进入、退出、替换等迁移关系。

## 迁移

### CALL-RETURN

* CALL：呼出新的View，新的View会被放置在栈顶，发起调用的View会被覆盖；
* RETURN：返回前一View，被覆盖的View重新进行显示。只有被CALL出来的View才能调用RETURN。

CALL-RETURN迁移时的Stack变化以及View回调函数执行情况如下：



图 ‑1 CALL-RETURN迁移

### CHANGE

* CHANGE：一个View替换为另一个View。

CHANGE时Stack变化以及View回调函数执行情况如下：



图 ‑2 CHANGE迁移

## 回调

用于在不同的迁移时进行必要的处理。具体的View可以根据需要覆盖（override）相应的回调函数：

* onCreate
* onEnter
* onHide
* onShow
* onExit

## 注册

为了使用ID来进行View迁移操作，需要根据View ID将View对象指针注册到索引表中。

## API

### regView

|  |  |  |
| --- | --- | --- |
| API名称 | regView | |
| 原型声明 | void regView(void \*\_self, int vid, View\_t\* view) | |
| 功能描述 | 注册View | |
| 参数(in) | \_self | self指针 |
| 参数(in) | vid | View ID。View ID不能超过STACK\_SIZE-1。 |
| 参数(in) | View | View对象指针 |
| 返回值 | - | - |
| 注意事项 | 所有迁移API都必须在注册后才能调用，否则程序会产生异常 | |

### setInPara

|  |  |  |
| --- | --- | --- |
| API名称 | setInPara | |
| 原型声明 | void setInPara(void \*\_self, int vid, Transfer\_t\* trans) | |
| 功能描述 | 设置迁移参数 | |
| 参数(in) | \_self | self指针 |
| 参数(in) | vid | View ID |
| 参数(in) | trans | 迁移参数 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### push1stView

|  |  |  |
| --- | --- | --- |
| API名称 | push1stView | |
| 原型声明 | void push1stView(void \*\_self, int vid) | |
| 功能描述 | 第一个View入栈API | |
| 参数(in) | \_self | self指针 |
| 参数(in) | vid | View ID |
| 返回值 | - | - |
| 注意事项 | 仅供App框架调用 | |

### callView

|  |  |  |
| --- | --- | --- |
| API名称 | callView | |
| 原型声明 | void callView(void \*\_self, int vid) | |
| 功能描述 | 呼出新View | |
| 参数(in) | \_self | self指针 |
| 参数(in) | vid | View ID |
| 返回值 | - | - |
| 注意事项 | 无 | |

### returnView

|  |  |  |
| --- | --- | --- |
| API名称 | returnView | |
| 原型声明 | void returnView(void \*\_self) | |
| 功能描述 | View退出 | |
| 参数(in) | \_self | self指针 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### changeView

|  |  |  |
| --- | --- | --- |
| API名称 | changeView | |
| 原型声明 | void changeView(void \*\_self, int vid) | |
| 功能描述 | 替换为新View | |
| 参数(in) | \_self | self指针 |
| 参数(in) | vid | View ID |
| 返回值 | - | - |
| 注意事项 | 无 | |

### getTopView

|  |  |  |
| --- | --- | --- |
| API名称 | getTopView | |
| 原型声明 | View\_t\* getTopView(void \*\_self) | |
| 功能描述 | 取得栈顶View（当前显示的View）指针 | |
| 参数(in) | \_self | self指针 |
| 返回值 | View\_t\* | 栈顶View指针 |
| 注意事项 | 无 | |

### transfer

|  |  |  |
| --- | --- | --- |
| API名称 | transfer | |
| 原型声明 | View\_t\* transfer (void \*\_self) | |
| 功能描述 | View迁移处理 | |
| 参数(in) | \_self | self指针 |
| 返回值 | View\_t\* | 栈顶View指针 |
| 注意事项 | 无 | |

# Graphic库

Graphic Lib实现基本的描画处理，包括图片、文本、矩形等。另外，也实现图片资源的管理。

## Resource

图片资源采用数组进行存储。图片信息包括图片数据地址、宽度、高度。图片像素为32bit（RGBA）。



图 ‑1 图片资源存储

由于用到的图片资源相对较少，所以在APP启动时加载所有的图片资源到内存中。

## Graphic API

### drawImage

|  |  |  |
| --- | --- | --- |
| API名称 | drawImage | |
| 原型声明 | void drawImage(double x, double y, int id) | |
| 功能描述 | 图片描画API | |
| 参数(in) | x | 图片左上角X坐标 |
| 参数(in) | y | 图片左上角Y坐标 |
| 参数(in) | id | 图片ID |
| 返回值 | - | - |
| 注意事项 | 无 | |

### drawImage2

|  |  |  |
| --- | --- | --- |
| API名称 | drawImage2 | |
| 原型声明 | void drawImage2(double x, double y, int id, int a) | |
| 功能描述 | 图片描画API（指定透明度） | |
| 参数(in) | x | 图片左上角X坐标 |
| 参数(in) | y | 图片左上角Y坐标 |
| 参数(in) | id | 图片ID |
| 参数(in) | a | 透明度，0～255。值越小透明度越大。 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### drawText

|  |  |  |
| --- | --- | --- |
| API名称 | drawText | |
| 原型声明 | void drawText(double x, double y, const char \*text) | |
| 功能描述 | 字符串描画API | |
| 参数(in) | x | X坐标 |
| 参数(in) | y | Y坐标 |
| 参数(in) | text | 字符串 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### drawText2

|  |  |  |
| --- | --- | --- |
| API名称 | drawText2 | |
| 原型声明 | void drawText(double x, double y, const char \*text, UIColor color) | |
| 功能描述 | 使用指定颜色字符串描画API | |
| 参数(in) | x | X坐标 |
| 参数(in) | y | Y坐标 |
| 参数(in) | text | 字符串 |
| 参数(in) | color | 颜色 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### drawTextC

|  |  |  |
| --- | --- | --- |
| API名称 | drawTextC | |
| 原型声明 | void drawTextC(double x, double y, const char \*text) | |
| 功能描述 | 字符串居中描画API | |
| 参数(in) | x | 字符串中心X坐标 |
| 参数(in) | y | Y坐标 |
| 参数(in) | text | 字符串 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### drawTextR

|  |  |  |
| --- | --- | --- |
| API名称 | drawTextR | |
| 原型声明 | void drawTextR(double x, double y, const char \*text) | |
| 功能描述 | 字符串靠右描画API | |
| 参数(in) | x | 字符串末尾X坐标 |
| 参数(in) | y | Y坐标 |
| 参数(in) | text | 字符串 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### getTextWidth

|  |  |  |
| --- | --- | --- |
| API名称 | getTextWidth | |
| 原型声明 | double getTextWidth(const char \*text) | |
| 功能描述 | 字符串宽度取得API | |
| 参数(in) | text | 字符串 |
| 返回值 | 字符串宽度 | 宽度的单位为像素 |
| 注意事项 | 无 | |

### drawRectFill

|  |  |  |
| --- | --- | --- |
| API名称 | drawRectFill | |
| 原型声明 | void drawRectFill (double x1, double y1, double x2, double y2, UIColor color) | |
| 功能描述 | 矩形填充API（颜色不能包含alpha通道，描画速度较快） | |
| 参数(in) | x1 | 左上角X坐标 |
| 参数(in) | y1 | 左上角Y坐标 |
| 参数(in) | x2 | 右下角X坐标 |
| 参数(in) | y2 | 右下角Y坐标 |
| 参数(in) | color | 颜色，不能包含alpha通道 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### drawRectFill2

|  |  |  |
| --- | --- | --- |
| API名称 | drawRectFill2 | |
| 原型声明 | void drawRectFill2 (double x1, double y1, double x2, double y2, UIColor color) | |
| 功能描述 | 矩形填充API（颜色可以包含alpha通道） | |
| 参数(in) | x1 | 左上角X坐标 |
| 参数(in) | y1 | 左上角Y坐标 |
| 参数(in) | x2 | 右下角X坐标 |
| 参数(in) | y2 | 右下角Y坐标 |
| 参数(in) | color | 颜色，可以包含alpha通道 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### drawRect

|  |  |  |
| --- | --- | --- |
| API名称 | drawRect | |
| 原型声明 | void drawRect (double x1, double y1, double x2, double y2, UIColor color) | |
| 功能描述 | 矩形描画API（无填充） | |
| 参数(in) | x1 | 左上角X坐标 |
| 参数(in) | y1 | 左上角Y坐标 |
| 参数(in) | x2 | 右下角X坐标 |
| 参数(in) | y2 | 右下角Y坐标 |
| 参数(in) | color | 颜色，可以包含alpha通道 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### allocResArray

|  |  |  |
| --- | --- | --- |
| API名称 | allocResArray | |
| 原型声明 | void allocResArray (int total) | |
| 功能描述 | 图片资源数组分配API | |
| 参数(in) | total | 图片资源总数 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### freeResArray

|  |  |  |
| --- | --- | --- |
| API名称 | freeResArray | |
| 原型声明 | void freeResArray (void) | |
| 功能描述 | 图片资源数组释放API，包括图片资源也会被释放 | |
| 返回值 | - | - |
| 注意事项 | 无 | |

### addImage

|  |  |  |
| --- | --- | --- |
| API名称 | addImage | |
| 原型声明 | void addImage(int index, const char \*fileName) | |
| 功能描述 | 图片资源加载API | |
| 参数(in) | index | 图片资源索引 |
| 参数(in) | fileName | 文件名 |
| 返回值 | - | - |
| 注意事项 | 需要在调用allocResArray才能调用 | |

### regVram

|  |  |  |
| --- | --- | --- |
| API名称 | regVram | |
| 原型声明 | void regVram(unsigned char \*vram, int w, int h) | |
| 功能描述 | VRAM信息注册API | |
| 参数(in) | vram | VRAM地址 |
| 参数(in) | w | 屏幕宽度 |
| 参数(in) | h | 屏幕高度 |
| 返回值 | - | - |
| 注意事项 | VRAM存储空间大小必须是3\*w\*h字节 | |

## Color API

### RGBA

|  |  |  |
| --- | --- | --- |
| API名称 | RGBA | |
| 原型声明 | UIColor RGBA(int r, int g, int b, int a) | |
| 功能描述 | 根据RGB以及Alpha设置颜色值 | |
| 参数(in) | r | Red |
| 参数(in) | g | Green |
| 参数(in) | b | Blue |
| 参数(in) | a | Alpha |
| 返回值 | UIColor | 颜色值 |
| 注意事项 | 无 | |

### RGB

|  |  |  |
| --- | --- | --- |
| API名称 | RGB | |
| 原型声明 | UIColor RGB(int r, int g, int b) | |
| 功能描述 | 根据RGB设置颜色值（Alpha默认为255） | |
| 参数(in) | r | Red |
| 参数(in) | g | Green |
| 参数(in) | b | Blue |
| 返回值 | UIColor | 颜色值 |
| 注意事项 | 无 | |

### BGR

|  |  |  |
| --- | --- | --- |
| API名称 | BGR | |
| 原型声明 | UIColor BGR(void) | |
| 功能描述 | 背景色取得API | |
| 返回值 | UIColor | 背景色 |
| 注意事项 | 无 | |

### setBGR

|  |  |  |
| --- | --- | --- |
| API名称 | setBGR | |
| 原型声明 | void setBGR (int r, int g, int b) | |
| 功能描述 | 设置背景色（Alpha默认为255） | |
| 参数(in) | r | Red |
| 参数(in) | g | Green |
| 参数(in) | b | Blue |
| 返回值 | - | - |
| 注意事项 | 无 | |

# Game Core模块

## 模块构成



图 ‑1 Game Core模块框图

Game Core实现主要分以下几个部分：

### 启动/终止API

游戏启动、中止相关处理。

#### game\_start

|  |  |  |
| --- | --- | --- |
| API名称 | game\_start | |
| 原型声明 | void game\_start(int index) | |
| 功能描述 | 启动游戏 | |
| 参数(in) | index | 关卡索引，范围0~35 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_end

|  |  |  |
| --- | --- | --- |
| API名称 | game\_end | |
| 原型声明 | void game\_end(void) | |
| 功能描述 | 中止游戏 | |
| 返回值 | - | - |
| 注意事项 | 无 | |

### 界面操作API

通常情况下，点击战场时，先调用game\_get\_object取得操作对象，之后再调用对应的API来进行操作。

#### game\_get\_object

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_object | |
| 原型声明 | game\_get\_object(int x, int y) | |
| 功能描述 | 取得指定位置的操作对象 | |
| 参数(in) | x | x坐标 |
| 参数(in) | y | y坐标 |
| 返回值 | int | 操作对象，详细如下：  typedef enum{  OPER\_OBJ\_ROAD = 0,//道路  OPER\_OBJ\_LAND,//土地  OPER\_OBJ\_PLANT,//庄稼  OPER\_OBJ\_PLANT\_MATURE,//成熟庄稼  OPER\_OBJ\_MINE,//地雷  OPER\_OBJ\_MINE\_LB,//路边炸弹  OPER\_OBJ\_MINE\_REMOTE,//遥控器  OPER\_OBJ\_MINE\_REMOTE2,//敌人射程内的遥控器  OPER\_OBJ\_MINE\_TIMER,//定时起爆器  OPER\_OBJ\_ROAD\_SIDE,//路边  OPER\_OBJ\_FOREST,//森林  OPER\_OBJ\_MINEGUN,//地雷炮  OPER\_OBJ\_MINEGUN\_NO\_FIRE,//待装填的地雷炮  OPER\_OBJ\_NONE,//无操作对象  OPER\_OBJ\_MAX  }operate\_obj\_t; |
| 注意事项 | 无 | |

#### game\_add\_plug

|  |  |  |
| --- | --- | --- |
| API名称 | game\_add\_plug | |
| 原型声明 | void game\_add\_plug(int x, int y, int plug) | |
| 功能描述 | 为路边炸弹添加PLUG | |
| 参数(in) | x | x坐标 |
| 参数(in) | y | y坐标 |
| 参数(in) | plug | PLUG\_TIMER：定时起爆器  PLUG\_REMOTE：遥控起爆器 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_add\_plant

|  |  |  |
| --- | --- | --- |
| API名称 | game\_add\_plant | |
| 原型声明 | void game\_add\_plant(int x, int y) | |
| 功能描述 | 种植庄稼 | |
| 参数(in) | x | x坐标 |
| 参数(in) | y | y坐标 |
| 返回值 | - | - |
| 注意事项 | (x,y)处必须是土地（FIELD\_LAND）。 | |

#### game\_add\_landmine

|  |  |  |
| --- | --- | --- |
| API名称 | game\_add\_landmine | |
| 原型声明 | void game\_add\_landmine(int x, int y, int id) | |
| 功能描述 | 部署地雷 | |
| 参数(in) | x | x坐标 |
| 参数(in) | y | y坐标 |
| 返回值 | - | - |
| 注意事项 | (x,y)处必须是道路（FIELD\_ROAD）或路边的土地（FIELD\_LAND）。 | |

#### game\_add\_minegun

|  |  |  |
| --- | --- | --- |
| API名称 | game\_add\_minegun | |
| 原型声明 | void game\_add\_minegun(int x, int y) | |
| 功能描述 | 部署地雷炮 | |
| 参数(in) | x | x坐标 |
| 参数(in) | y | y坐标 |
| 返回值 | - | - |
| 注意事项 | (x,y)处必须是森林（FIELD\_FOREST）或土地（FIELD\_LAND）。 | |

#### game\_harvest\_plant

|  |  |  |
| --- | --- | --- |
| API名称 | game\_harvest\_plant | |
| 原型声明 | void game\_harvest\_plant(int x, int y) | |
| 功能描述 | 收割庄稼 | |
| 参数(in) | x | x坐标 |
| 参数(in) | y | y坐标 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_dig\_landmine

|  |  |  |
| --- | --- | --- |
| API名称 | game\_dig\_landmine | |
| 原型声明 | void game\_dig\_landmine(int x, int y) | |
| 功能描述 | 挖出（回收）地雷 | |
| 参数(in) | x | x坐标 |
| 参数(in) | y | y坐标 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_dig\_minegun

|  |  |  |
| --- | --- | --- |
| API名称 | game\_dig\_minegun | |
| 原型声明 | void game\_dig\_minegun(int x, int y) | |
| 功能描述 | 回收地雷炮 | |
| 参数(in) | x | x坐标 |
| 参数(in) | y | y坐标 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_watering\_plant

|  |  |  |
| --- | --- | --- |
| API名称 | game\_watering\_plant | |
| 原型声明 | void game\_watering\_plant(int x, int y) | |
| 功能描述 | 给庄稼浇水 | |
| 参数(in) | x | x坐标 |
| 参数(in) | y | y坐标 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_explose\_landmine

|  |  |  |
| --- | --- | --- |
| API名称 | game\_explose\_landmine | |
| 原型声明 | void game\_explose\_landmine(int x, int y) | |
| 功能描述 | 引爆地雷 | |
| 参数(in) | x | x坐标 |
| 参数(in) | y | y坐标 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_minegun\_fire

|  |  |  |
| --- | --- | --- |
| API名称 | game\_minegun\_fire | |
| 原型声明 | void game\_minegun\_fire(int x, int y, int tx, int ty) | |
| 功能描述 | 地雷炮开火 | |
| 参数(in) | x | 地雷炮x坐标 |
| 参数(in) | y | 地雷炮y坐标 |
| 参数(in) | tx | 目标x坐标 |
| 参数(in) | ty | 目标y坐标 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### 数据取得API

数据取得API主要用于显示数据的读取，通常用于游戏画面的显示。

#### game\_get\_enemy\_list

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_enemy\_list | |
| 原型声明 | void game\_get\_enemy\_list(dlist\_t \*\*list) | |
| 功能描述 | 取得敌人列表 | |
| 参数(out) | list | 链表头指针地址 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_dead\_enemy\_list

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_dead\_enemy\_list | |
| 原型声明 | void game\_get\_dead\_enemy\_list(dlist\_t \*\*list) | |
| 功能描述 | 取得死亡敌人列表（用于点闪显示） | |
| 参数(out) | list | 链表头指针地址 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_bullet\_list

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_bullet\_list | |
| 原型声明 | void game\_get\_bullet\_list(dlist\_t \*\*list) | |
| 功能描述 | 取得子弹/炮弹列表 | |
| 参数(out) | list | 链表头指针地址 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_landmine\_list

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_landmine\_list | |
| 原型声明 | void game\_get\_landmine\_list(dlist\_t \*\*list) | |
| 功能描述 | 取得地雷列表 | |
| 参数(out) | list | 链表头指针地址 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_exploded\_mine\_list

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_exploded\_mine\_list | |
| 原型声明 | void game\_get\_exploded\_mine\_list(dlist\_t \*\*list) | |
| 功能描述 | 取得爆炸的地雷列表（用于爆炸效果显示） | |
| 参数(out) | list | 链表头指针地址 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_minegun\_list

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_minegun\_list | |
| 原型声明 | void game\_get\_minegun\_list(dlist\_t \*\*list) | |
| 功能描述 | 取得地雷炮列表 | |
| 参数(out) | list | 链表头指针地址 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_plant\_list

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_plant\_list | |
| 原型声明 | void game\_get\_plant\_list(dlist\_t \*\*list) | |
| 功能描述 | 取得庄稼列表 | |
| 参数(out) | list | 链表头指针地址 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_house\_list

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_house\_list | |
| 原型声明 | void game\_get\_house\_list(dlist\_t \*\*list) | |
| 功能描述 | 取得房屋列表 | |
| 参数(out) | list | 链表头指针地址 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_map\_index

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_map\_index | |
| 原型声明 | int game\_get\_map\_index(void) | |
| 功能描述 | 取得关卡索引 | |
| 返回值 | int | 关卡索引，范围0～MAP\_TOAL-1。 |
| 注意事项 | 无 | |

#### game\_get\_map\_data

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_map\_data | |
| 原型声明 | void game\_get\_map\_data(int map[MAP\_H][MAP\_W]) | |
| 功能描述 | 取得地图数据 | |
| 参数(out) | map | 地图数据存储数组地址 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_direction\_data

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_direction\_data | |
| 原型声明 | void game\_get\_direction\_data(int direction[MAP\_H][MAP\_W]) | |
| 功能描述 | 取得移动方向数据 | |
| 参数(out) | direction | 移动方向数据存储地址，移动方向定义如下：  typedef enum{  MOVE\_U = -2, /\*\*< move up \*/  MOVE\_L = -1, /\*\*< move left \*/  MOVE\_R = 1, /\*\*< move right \*/  MOVE\_D = 2 /\*\*< move down \*/  } move\_dir\_t; |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_money

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_money | |
| 原型声明 | int game\_get\_money(void) | |
| 功能描述 | 取得金钱数量 | |
| 返回值 | int | 金钱数量 |
| 注意事项 | 无 | |

#### game\_get\_over\_flag

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_over\_flag | |
| 原型声明 | int game\_get\_over\_flag(void) | |
| 功能描述 | 取得游戏结果标志 | |
| 返回值 | int | 游戏结果，定义如下：  typedef enum{  OVER\_NONE = 0,//未结束  OVER\_WIN,//胜利  OVER\_LOSE,//失败  OVER\_MAX  } over\_flag\_t; |
| 注意事项 | 无 | |

#### game\_is\_all\_house\_ok

|  |  |  |
| --- | --- | --- |
| API名称 | game\_is\_all\_house\_ok | |
| 原型声明 | int game\_is\_all\_house\_ok(void) | |
| 功能描述 | 判断是否所有的房屋没有被破坏 | |
| 返回值 | int | 0：有房屋损坏  1：所有房屋都没有损坏 |
| 注意事项 | 无 | |

#### game\_is\_plants\_star

|  |  |  |
| --- | --- | --- |
| API名称 | game\_is\_plants\_star | |
| 原型声明 | int game\_is\_plants\_star(void) | |
| 功能描述 | 判断是否取得plant star | |
| 返回值 | int | 0：没有取得plant star  1：取得plant star |
| 注意事项 | 无 | |

#### game\_get\_score

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_score | |
| 原型声明 | int game\_get\_score(void) | |
| 功能描述 | 取得分数 | |
| 返回值 | int | 分数 |
| 注意事项 | 无 | |

#### game\_get\_enemy\_total

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_enemy\_total | |
| 原型声明 | int game\_get\_enemy\_total(void) | |
| 功能描述 | 取得敌人总数 | |
| 返回值 | int | 敌人总数 |
| 注意事项 | 无 | |

#### game\_get\_enemy\_killed

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_enemy\_killed | |
| 原型声明 | int game\_get\_enemy\_killed(void) | |
| 功能描述 | 取得已消灭敌人数量 | |
| 返回值 | int | 已消灭敌人数 |
| 注意事项 | 无 | |

#### game\_get\_entrance

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_entrance | |
| 原型声明 | void game\_get\_entrance(point\_t \*entry, int \*total) | |
| 功能描述 | 取得地图入口 | |
| 参数(out) | entry | 入口坐标数组，需要4个元素 |
| 参数(out) | total | 入口数量，范围1～4 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_stars

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_stars | |
| 原型声明 | void game\_get\_stars(star\_data\_t stars[GAME\_STAR\_TOTAL]) | |
| 功能描述 | 取得荣誉星数据 | |
| 参数(out) | stars | 荣誉星数据  typedef struct{  int gotten;//是否取得  int progress;//当前完成值  int target;//目标  } star\_data\_t; |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_name

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_name | |
| 原型声明 | void game\_get\_name(int index, char \*name) | |
| 功能描述 | 取得关卡名称 | |
| 参数(in) | index | 关卡索引 |
| 参数(out) | name | 关卡名称 |
| 返回值 | - | - |
| 注意事项 | 无 | |

#### game\_get\_danger\_range

|  |  |  |
| --- | --- | --- |
| API名称 | game\_get\_danger\_range | |
| 原型声明 | void game\_get\_danger\_range(int map[MAP\_H][MAP\_W]) | |
| 功能描述 | 取得危险区域（敌人射程范围） | |
| 参数(out) | map | 危险区域  0：安全/1：危险 |
| 返回值 | - | - |
| 注意事项 | 无 | |

### 周期处理

周期处理流程如下：

1. 敌人队列移动Enemy Move：

由对头开始向前移动一格；

被炸损坏的敌人延迟一次移动；

前方敌人未移动，后方也无法移动；

攻击力强敌人会和攻击力弱敌人交换位置；

1. 敌人部署Enemy Dispose：

根据队列时间间隔进入地图相应入口；

1. 踩雷爆炸检测Explosion Check：

根据地雷引爆规则，判定地雷是否爆炸；

1. 庄稼生长Plant Grow：

根据庄稼水量以及时间进行状态迁移；

1. 子弹移动Bullet Move：

子弹飞行处理；

1. 开火检查Fire Check：

判断敌人是否要开火；

1. 子弹/炮弹Bullet Hit Check：

子弹/炮弹爆炸以及攻击处理；

1. 游戏状态检测Over Check：

判断游戏是否结束以及结果。



图 ‑2 Game周期处理

|  |  |  |
| --- | --- | --- |
| API名称 | game\_tick | |
| 原型声明 | void game\_tick (void) | |
| 功能描述 | 游戏周期处理 | |
| 返回值 | - | - |
| 注意事项 | 需要以10ms周期进行调用。 | |

## 双向链表

双向链表用于存储游戏中各种数据，包括敌人队列、地雷、庄稼、房子、子弹等。

### 数据结构

为了实现通用的数据存储，在节点中只存储数据地址，而不关心具体的数据类型。



图 ‑3 双向链表

### 通用算法

#### 实现

* 在游戏的实现中，会涉及各种条件的插入、删除、查找操作。为了实现多种匹配条件，采用callback函数的形式。下面以find函数来说明实现的过程：
* 匹配函数：

typedef int (\*cmp\_func\_t)(void \*p1, void \*p2);

1. 调用find方法时，调用者传入参考数据data和条件匹配函数func；
2. find函数中遍历链表时，调用匹配函数func。链表节点数据作为参数p1，参考数据作为参数p2；
3. 匹配函数根据需要对两个数据进行比较，返回是否匹配；
4. find函数根据匹配结果判断是否结束遍历。

通过参考数据data和匹配函数func，可以灵活的实现各种条件的查找。另外，当匹配条件仅判断链表节点数据时，data参数可以传递NULL。此时传入的回调函数func不能使用p2参数。



图 ‑4 查找

* 查找算法流程：



图 ‑5 查找算法

#### 匹配函数示例

* 查找坐标为(x, y)的敌人：

int enemy\_xy\_cmp(void \*v1, void \*v2)

{

int ret = 1;

enemy\_t \*p1 = v1;

enemy\_t \*p2 = v2;

if((p1->x == p2->x)&&(p1->y == p2->y)){

ret = 0;

}

return ret;

}

enemy.x = landmine->x;

enemy.y = landmine->y;

enemy2 = dlist\_find(enemy\_list, &enemy, enemy\_xy\_cmp);

* 查找死亡的敌人：

只根据链表节点数据中life值匹配，因此参考data传入NULL。

int enemy\_dead\_cmp(void \*v1, void \*v2)

{

int ret = 1;

enemy\_t \*p1 = v1;

if(p1->life <= 0){

ret = 0;

}

return ret;

}

enemy2 = dlist\_find(enemy\_list, NULL, enemy\_dead\_cmp);

### API

#### dlist\_push\_front

|  |  |  |
| --- | --- | --- |
| API名称 | dlist\_push\_front | |
| 原型声明 | int dlist\_push\_front(dlist\_t \*\*dlist, void \*data) | |
| 功能描述 | 向链表头部追加节点 | |
| 参数(in/out) | dlist | 链表头指针地址 |
| 参数(in) | data | 节点数据地址，必须是动态分配的空间 |
| 返回值 | int | 0：失败/1：成功 |
| 注意事项 | 无 | |

#### dlist\_push\_back

|  |  |  |
| --- | --- | --- |
| API名称 | dlist\_push\_back | |
| 原型声明 | int dlist\_push\_back(dlist\_t \*\*dlist, void \*data) | |
| 功能描述 | 向链表尾部追加节点 | |
| 参数(in/out) | dlist | 链表头指针地址 |
| 参数(in) | data | 节点数据地址，必须是动态分配的空间 |
| 返回值 | int | 0：失败/1：成功 |
| 注意事项 | 无 | |

#### dlist\_ins

|  |  |  |
| --- | --- | --- |
| API名称 | dlist\_ins | |
| 原型声明 | int dlist\_ins(dlist\_t \*\*dlist, void \*data, void \*data2, cmp\_func\_t func) | |
| 功能描述 | 向符合条件的节点前插入新节点 | |
| 参数(in/out) | dlist | 链表头指针地址 |
| 参数(in) | data | 条件匹配数据，用于传递给回调函数func的第二参数 |
| 参数(in) | data2 | 要插入的节点数据地址，必须是动态分配的空间 |
| 参数(in) | func | 条件匹配回调函数 |
| 返回值 | int | 0：失败/1：成功 |
| 注意事项 | 无 | |

#### dlist\_del

|  |  |  |
| --- | --- | --- |
| API名称 | dlist\_del | |
| 原型声明 | int dlist\_del(dlist\_t \*\*dlist, void \*data, cmp\_func\_t func) | |
| 功能描述 | 删除符合条件第一个节点 | |
| 参数(in/out) | dlist | 链表头指针地址 |
| 参数(in) | data | 条件匹配数据，用于传递给回调函数func的第二参数 |
| 参数(in) | func | 条件匹配回调函数 |
| 返回值 | int | 0：失败/1：成功 |
| 注意事项 | 无 | |

#### dlist\_find

|  |  |  |
| --- | --- | --- |
| API名称 | dlist\_find | |
| 原型声明 | void \*dlist\_find(dlist\_t \*dlist, void \*data, cmp\_func\_t func) | |
| 功能描述 | 检索符合条件的数据节点 | |
| 参数(in/out) | dlist | 链表头指针 |
| 参数(in) | data | 条件匹配数据，用于传递给回调函数func的第二参数 |
| 参数(in) | func | 条件匹配回调函数 |
| 返回值 | void \* | NULL：没有检索到/非NULL：检索到的数据地址 |
| 注意事项 | 无 | |

#### dlist\_get\_size

|  |  |  |
| --- | --- | --- |
| API名称 | dlist\_get\_size | |
| 原型声明 | int dlist\_get\_size(dlist\_t \*dlist) | |
| 功能描述 | 取得链表长度 | |
| 参数(in/out) | dlist | 链表头指针 |
| 返回值 | int | 链表长度 |
| 注意事项 | 无 | |

#### dlist\_clear

|  |  |  |
| --- | --- | --- |
| API名称 | dlist\_clear | |
| 原型声明 | void dlist\_clear(dlist\_t \*\*dlist) | |
| 功能描述 | 清除链表所有节点，包括释放节点数据 | |
| 参数(in/out) | dlist | 链表头指针地址 |
| 返回值 | - | - |
| 注意事项 | 无 | |

## 数据结构

### 地图

采用14\*9的数组来实现，不同的值代表不同的元素。

表 ‑1 地图元素

|  |  |  |
| --- | --- | --- |
| 枚举 | 值 | 描述 |
| FIELD\_ROAD | 0 | 道路 |
| FIELD\_LAND | 1 | 土地 |
| FIELD\_FOREST | 2 | 森林 |
| FIELD\_HOUSE | 3 | 房屋 |
| FIELD\_WATER | 4 | 水面 |
| FIELD\_ENTRANCE | 5 | 道路入口，最少有1个入口，最多4个 |
| FIELD\_END | 6 | 道路终点 |

* 地图定义原则：

1. 可以有1~4个入口；
2. 可以有1～2个道路终点；
3. 每个入口只能对应一个终点，不允许一个入口可以到达两个终点的情况；
4. 每个入口到终点只有唯一路径。也就是即使道路有交叉点，交叉点到终点只有唯一路径。

### 队列

队列最大长度50。队列元素包括Enemy id和时间间隔。

表 ‑2 队列元素

|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| id | enemy\_id\_t | 敌人类型ID，范围如下：  typedef enum{  ENEMY\_SB = 0, /\*\*<士兵\*/  ENEMY\_GB, /\*\*<工兵\*/  ENEMY\_QC, /\*\*<汽车\*/  ENEMY\_ZJ, /\*\*<装甲车\*/  ENEMY\_TK, /\*\*<坦克\*/  ENEMY\_MAX  } enemy\_id\_t; |
| interval | int | 前后时间间隔 |

### 敌人

表 ‑3 敌人信息

|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| id | enemy\_id\_t | 敌人类型ID，范围如下：  typedef enum{  ENEMY\_SB = 0, /\*\*<士兵\*/  ENEMY\_GB, /\*\*<工兵\*/  ENEMY\_QC, /\*\*<汽车\*/  ENEMY\_ZJ, /\*\*<装甲车\*/  ENEMY\_TK, /\*\*<坦克\*/  ENEMY\_MAX  } enemy\_id\_t; |
| x | int | x坐标 |
| y | int | y坐标 |
| life | int | 生命值 |
| time | int | 时间，用途如下：  1)进入入口的时间（相对于前一敌人的间隔）；  2)死亡敌人的显示时间。 |
| delay | int | 破坏延迟，延迟的移动周期数 |
| firing\_interval | int | 开火间隔，实现不同的开火间隔 |
| moved | int | 是否移动标志，0：未移动/1：已移动 |
| direction | enemy\_dir\_t | 移动方向，详细如下：  typedef enum{  ENEMY\_DIR\_R = 0,/\*\*<右\*/  ENEMY\_DIR\_U, /\*\*<上\*/  ENEMY\_DIR\_L, /\*\*<左\*/  ENEMY\_DIR\_D, /\*\*<下\*/  ENEMY\_DIR\_MAX  } enemy\_dir\_t; |
| entry | int | 进入战场的入口 |

### 地雷

表 ‑4 地雷信息

|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| id | landmine\_id\_t | 地雷类型ID，定义如下：  typedef enum{  MINE\_NM = 0, /\*\*<普通地雷\*/  MINE\_AE, /\*\*<反工兵地雷\*/  MINE\_RS, /\*\*<路边炸弹\*/  MINE\_AA, /\*\*<反装甲地雷\*/  MINE\_AT, /\*\*<反坦克地雷\*/  MINE\_MAX  } landmine\_id\_t; |
| x | int | x坐标 |
| y | int | y坐标 |
| plug | int | 插件，定义如下：  typedef enum{  PLUG\_NONE = 0, /\*\*<无\*/  PLUG\_TIMER, /\*\*<定时起爆器\*/  PLUG\_REMOTE, /\*\*<遥控起爆器\*/  PLUG\_MAX  } plug\_t; |
| time | int | 时间，用于定时器计时 |
| exploded | int | 是否已爆炸 |

### 庄稼

表 ‑5庄稼信息

|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| id | int | id(未使用) |
| x | int | x坐标 |
| y | int | y坐标 |
| state | plant\_status\_t | 庄稼状态，定义如下：  typedef enum{  PLANT\_STATE\_SEED, /\*\*<种子\*/  PLANT\_STATE\_SEEDLING,/\*\*<小苗\*/  PLANT\_STATE\_YOUNG, /\*\*<青苗\*/  PLANT\_STATE\_MATURE, /\*\*<成熟\*/  PLANT\_STATE\_MAX  } plant\_status\_t; |
| time | int | 时间，用于生长计时 |
| water | int | 水量。只有水量大于0时庄稼才生长。状态改变时，水量清0。需要浇水才能继续生长 |

* 庄稼状态迁移：



图 ‑6 庄稼状态迁移

### 房屋

表 ‑6 房屋信息

|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| id | int | id(未使用) |
| x | int | x坐标 |
| y | int | y坐标 |
| life | int | 生命值 |

### 子弹

表 ‑7 子弹信息

|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| id | bullet\_id\_t | 子弹类型，定义如下：  typedef enum{  BULLET\_BQ = 0, /\*\*<步枪/机枪\*/  BULLET\_ZJ, /\*\*<装甲车炮 \*/  BULLET\_TK, /\*\*<坦克炮\*/  BULLET\_MG, /\*\*<地雷炮\*/  BULLET\_MAX  } bullet\_id\_t; |
| x | int | x坐标 |
| y | int | y坐标 |
| ex | int | 目标x坐标 |
| ey | int | 目标y坐标 |
| time | int | 时间，用于显示爆炸效果 |
| state | int | 子弹状态，用于显示炸药包的不同图片 |

* 子弹飞行算法：

1. 当前坐标X坐标和目标点X坐标不同时，根据当前点坐标和目标点做坐标计算出飞行路线和x轴夹角的sin值和cos值，再计算步长STEP在x和y方向的分量，最后更新当前点坐标；



图 ‑7 子弹移动（非垂直）

1. 当前点X坐标和目标点X坐标相同时，直接更改Y坐标即可；



图 ‑8 子弹移动（垂直）

1. 当前坐标和目标点坐标距离小于STEP时，直接移动到目标点坐标。

### 地雷炮

表 ‑8 地雷炮信息

|  |  |  |
| --- | --- | --- |
| 成员变量 | 类型 | 描述 |
| id | minegun\_id\_t | 地雷炮类型，定义如下：  typedef enum{  MINEGUN\_0 = 0, /\*\*<土地上\*/  MINEGUN\_1, /\*\*<森里中\*/  MINEGUN\_MAX  } minegun\_id\_t; |
| x | int | x坐标 |
| y | int | y坐标 |
| life | int | 生命值 |
| time | int | 装填间隔 |
| exposed | int | 是否暴露，值如下：  0：在森里中未开火时为隐藏状态；  1：其余为暴露状态 |

## 算法

### 敌人队列移动

原始地图数据：其中0是道路，5是入口，6是终点。

{2,2,2,1,1,4,4,4,4,1,1,2,2,2},

{2,1,1,1,1,1,4,1,1,1,1,1,1,1},

{1,1,0,0,0,1,1,1,1,1,1,1,3,3},

{1,1,0,1,0,1,0,0,0,0,0,0,0,6},

{1,1,0,1,0,1,0,1,1,1,1,1,3,3},

{5,0,0,1,0,1,0,1,1,1,1,1,1,1},

{1,1,1,1,0,0,0,2,2,4,4,4,1,1},

{1,1,1,4,4,2,2,2,2,2,4,4,4,1},

{1,1,4,4,2,2,2,2,2,2,2,4,4,4},

转换后的移动方向数据：

{0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,R,R,D,0,0,0,0,0,0,0,0,0},

{0,0,U,0,D,0,R,R,R,R,R,R,R,6},

{0,0,U,0,D,0,U,0,0,0,0,0,0,0},

{R,R,U,0,D,0,U,0,0,0,0,0,0,0},

{0,0,0,0,R,R,U,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0},

{0,0,0,0,0,0,0,0,0,0,0,0,0,0},

* 地图-移动方向转换算法：

1. 将方向数据清0；
2. 从终点开始向左上右下四个方向查询地图，如果在(地图内)且是(入口或道路)且(未记录)方向，记录当前查询点方向，之后以当前查询点坐标递归调用；如果在地图外或不是入口或道路，则不进行任何操作；
3. 递归完所有道路和入口；
4. 如果是多个终点，则使用循环调用转换算法，分别从各个终点进行递归调用。



图 ‑9 地图-移动方向转换算法

//使用示例：

static map[MAP\_H][MAP\_W];

static direction[MAP\_H][MAP\_W];

memcpy(map, &all\_map\_data[index\*MAP\_H\*MAP\_W], sizeof(int)\*MAP\_W\*MAP\_H);

memcpy(direction, 0, sizeof(direction));

for(i = 0; i < end\_point\_total; i++){

direction[end\_point[i].y][end\_point[i].x] = 6;

}

for(i = 0; i < end\_point\_toal; i++){

fill\_move(end\_point[i].x, end\_point[i].y);

}

### 游戏难度

游戏难度是一个比较难以量化的值。就MineWar而言，影响难度的因素大概如下图：

* 对于正比因素而言，值越大则难度越大；
* 对于反比因素而言，值越大，难度越小。

虽然很难量化难度系数，但通过调整相关的值，可以改变难度。具体什么样的值是合适的，则需要通过实际的操作来验证。



图 ‑10 游戏难度因素

## 配置

实现游戏各种参数的动态调整，以调整游戏的难度和过关要点。

* 各种配置可以通过界面进行调整；
* 为了记录配置数据，需要保存到文件，并能在下次启动时自动加载；
* 文件打开失败时，需要使用默认设置值、地图以及队列数据。



图 ‑11 配置文件

### 基本数据

基本配置数据包括以下项目：

表 ‑9 基本配置数据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 数值ID | 最小值 | 最大值 | 调整步长 | 默认值 | 描述 |
| ENEMY\_DISPOSE\_CYCLE | 10 | 1000 | 10 | 20 | 部署周期 |
| ENEMY\_MOVE\_CYCLE | 500 | 10000 | 500 | 1000 | 移动周期 |
| BULLET\_MOVE\_CYCLE | 10 | 500 | 10 | 50 | 子弹移动周期 |
| PLANT\_GROW\_CYCLE | 1000 | 50000 | 1000 | 7000 | 庄稼生长周期 |
| PLUG\_TIMER\_CYCLE | 2000 | 20000 | 1000 | 10000 | 定时起爆时间 |
| INITIAL\_MONEY | 50 | 5000 | 50 | 1000 | 初始金钱 |
| DISPOSE\_OFFSET | 0 | 100 | 1 | 100 | 部署整体偏移 |
| PLANT\_PRICE | 5 | 100 | 5 | 5 | 庄稼价格 |
| PLANT\_HARVEST | 2 | 20 | 1 | 5 | 收获庄稼价格 |
| MINE\_PRICE\_NM | 5 | 200 | 5 | 5 | 普通地雷价格 |
| MINE\_PRICE\_AE | 5 | 200 | 5 | 10 | 反工兵地雷价格 |
| MINE\_PRICE\_RS | 5 | 200 | 5 | 20 | 路边炸弹价格 |
| MINE\_PRICE\_AA | 5 | 200 | 5 | 30 | 反装甲地雷价格 |
| MINE\_PRICE\_AT | 5 | 200 | 5 | 40 | 反坦克地雷价格 |
| PLUG\_PRICE\_TIMER | 5 | 200 | 5 | 40 | 定期起爆器价格 |
| PLUG\_PRICE\_REMOTE | 5 | 200 | 5 | 50 | 遥控起爆器价格 |
| GAME\_SPEED | 0 | 1 | 1 | 0 | 游戏速度 |
| LIMIT\_MONEY | 100 | 20000 | 100 | 1000 | 简单模式初始金钱 |
| QUEUE\_RANDOM | 0 | 1 | 1 | 0 | 随机队列 |

### 地图

需要能够通过界面编辑各个关卡的地图，编辑后的地图需要保存到文件并重新加载。

### 敌人队列

能够编辑敌人队列，包括敌人类型、时间间隔、进入地图的入口。编辑后的队列需要保存到文件并重新加载。

# UI实现

## App界面迁移

* 初始画面为Opening；
* Opening后显示Map画面，使用CHANGE迁移；
* Map画面到其它画面的迁移都是用CALL迁移，从其它画面返回使用RETURN迁移。



图 ‑1 应用界面迁移

## 游戏界面操作及状态迁移

### 状态迁移

* 游戏界面操作较为复杂，具体操作示例如下：



图 ‑2 游戏界面操作

* 抽象后的状态迁移：



图 ‑3 游戏界面操作状态迁移

表 ‑1 操作状态定义

|  |  |  |
| --- | --- | --- |
| 状态定义 | 描述 | 迁移描述 |
| POS\_SELECT | 位置选择状态，无菜单显示 | 如果工具栏需要切换，则切换到OPERATING\_SELECT状态；否则，按照工具栏当前选项进行动作 |
| OPERATING\_SELECT | 操作选择状态，显示菜单 | 如果点击工具栏，状态不变，切换工具栏选项，菜单显示不变；如果点击菜单以外区域工具栏或菜单或菜单需要切换，状态不变，切换菜单显示；点击AIM之外的菜单项，执行动作并迁移会POS\_SELECT状态；如果点击AIM，则进入AIM状态 |
| AIM | 地雷炮瞄准状态，显示地雷炮射程 | 点击射程内对选择位置开火，之后迁移回POS\_SELECT状态；点击射程外，迁移回POS\_SELECT状态，无其它动作 |

### 工具栏

工具栏分为两组：

* 道路工具栏：地雷部署工具，包括反工兵、普通、反装甲、反坦克地雷。默认选项是反工兵地雷；
* 其它工具栏：包括种子、镰刀、路边炸弹、地雷炮。默认选项是种子。

切换工具栏时，直接切换到默认选项。

### 操作对象及菜单项

表 ‑2 操作对象及菜单项

|  |  |  |
| --- | --- | --- |
| 操作对象 | 描述 | 菜单项 |
| ROAD | 道路 | 1. 当前工具栏是道路工具栏：   直接根据工具栏选项部署地雷  当前工具栏非地雷：   1. 切换工具栏，显示菜单反工兵、普通、反装甲、反坦克地雷 |
| LAND | 土地 | 1. 当前工具栏是道路工具栏：   切换工具栏，显示菜单种子、地雷炮   1. 当前工具栏上其它工具栏：   直接根据工具栏选项进行操作 |
| PLANT | 庄稼 | 无菜单项，如果当前工具栏项是镰刀，则进行收割；如果是其它，则进行浇水 |
| PLANT\_MATURE | 成熟的庄稼 | 无菜单项，直接进行收割 |
| MINE | 地雷 | 铁锹 |
| MINE\_LB | 路边炸弹 | 遥控器、铁锹 |
| MINE\_REMOTE | 在敌人射程内的遥控器 | 引爆 |
| MINE\_REMOTE2 | 遥控器 | 引爆、铁锹 |
| MINE\_TIMER | 定时起爆器 | 铁锹（未使用） |
| ROAD\_SIDE | 路边 | 1. 当前工具栏是道路工具栏：   切换工具栏，显示菜单庄稼、路边炸弹、地雷炮   1. 当前工具栏是其它工具栏：   直接根据工具栏选项进行操作 |
| FOREST | 森林 | 1. 当前工具栏是道路工具栏：   切换工具栏，显示菜单地雷炮   1. 当前工具栏是其它工具栏：   如果当前选项是地雷炮，直接部署地雷炮，不显示菜单。如果不是，则显示菜单 |
| MINEGUN | 地雷炮 | 开火、铁锹 |
| MINEGUN\_NO\_FIRE | 等待装填的地雷炮 | 铁锹 |
| NONE | 无操作对象 | 水面或者是在敌人射程内的道路、土地、森林、地雷、庄稼等除遥控引爆器和地雷炮以外的区域。点击无操作对象区会切换位置选择光标，取消菜单显示 |

## 游戏状态迁移

游戏状态划分如下：

* PLAYING：游戏中，可进行各种操作；
* PAUSE：暂停状态，不允许进行操作；
* OVER：游戏结束，显示游戏结果画面。



图 ‑4 游戏状态迁移

# 调试Log

Log用于跟踪程序运行情况，通常用于Bug调查、调试等。

## 类型

表 ‑1 Log类型

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 类型 | 级别 | 用途 | Debug版 | Release版 |
| ERROR | 1 | 错误信息 | 有 | 有 |
| WARNING | 2 | 警告信息 | 有 | 有 |
| TRACE | 3 | 跟踪信息 | 有 | 无 |
| INFO | 4 | 通常信息 | 有 | 无 |
| DEBUG | 5 | 调试信息 | 有 | 无 |

## 实现

采用不定参数的宏来实现log API。

#define \_LOGE(tag, format, ...) printf("["tag"/E] "format, ##\_\_VA\_ARGS\_\_)

在Release版中，可以方便的去掉log。

#define \_LOGE(tag, format, ...)

## API

|  |  |  |
| --- | --- | --- |
| API名称 | \_LOGE/\_LOGW/\_LOGT/\_LOGI/\_LOGD | |
| 原型声明 | #define \_LOGE(tag, format, ...) | |
| 功能描述 | 输出LOG(ERROR/INFO/TRACE/DEBUG/WARNING) | |
| 参数(in) | tag | 字符串，一般为模块名称或名称缩写 |
| 参数(in) | format | 格式字符串，类似printf |
| 参数(in) | ... | 不定参数，类似printf |
| 返回值 | - | - |
| 注意事项 | 无 | |