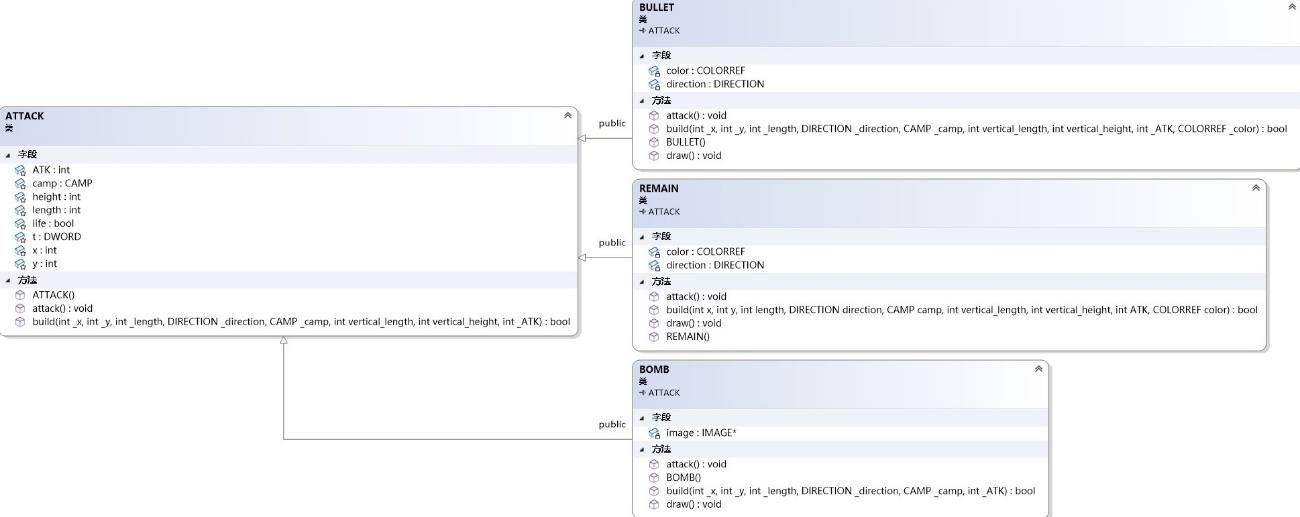
**设计说明书**

1. **类图**
   1. **组合**

直线右边的类的成员中有左边的类（或其派生类）的对象。标有（封装）的是结构体，包含左边类的若干派生类的对象。

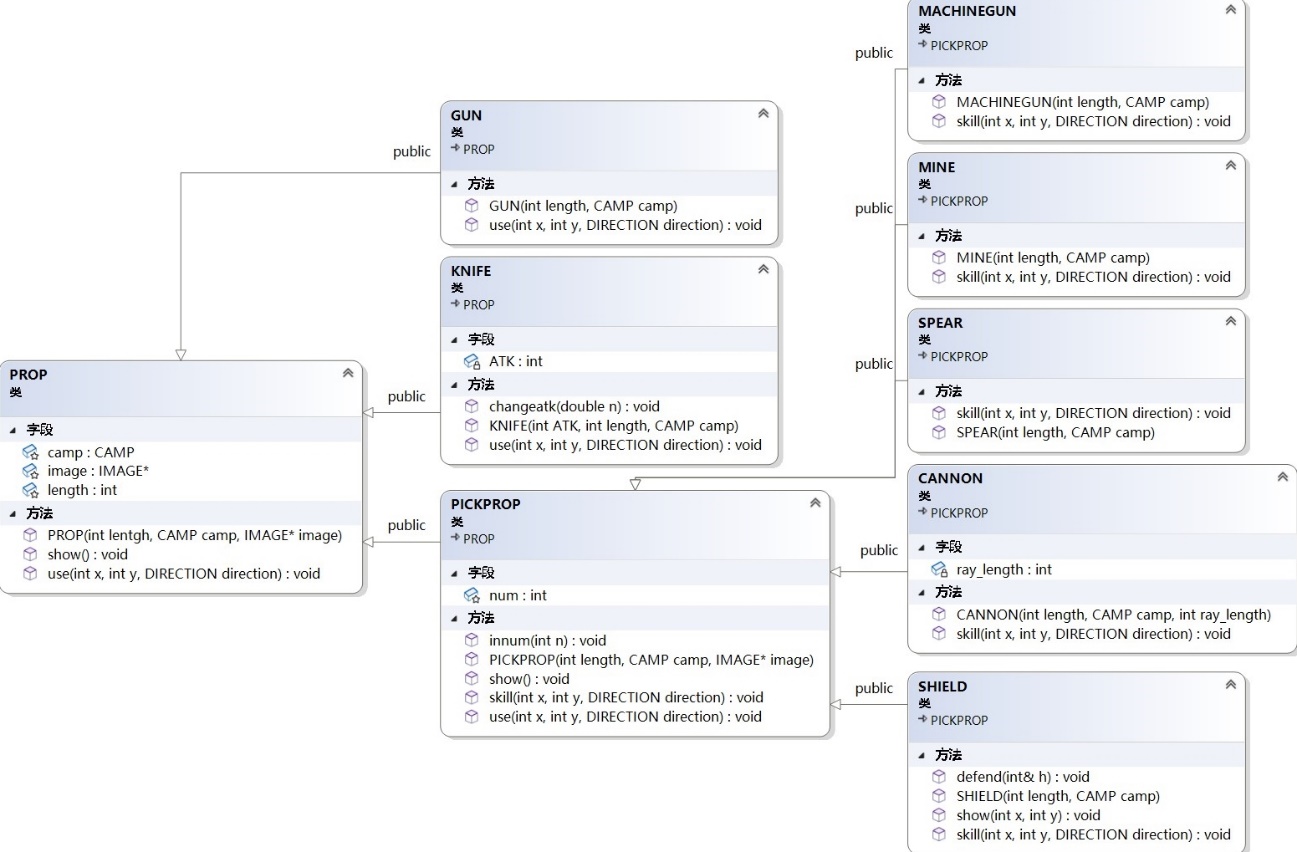
* 1. **继承(构造函数、析构函数、虚函数在派生类的实现不予赘述)**
     1. **ATTACK（攻击物）**



|  |  |  |  |
| --- | --- | --- | --- |
| **类名** | **数据成员名** | **数据类型** | **含义(用途)** |
| ATTACK | x | int | 位置坐标 |
| y | int | 位置坐标 |
| length | int | 水平方向长度 |
| Height | int | 垂直方向高度 |
| life | bool | 是否已被创建 |
| camp | CAMP | 阵营 |
| t | DEWORD | 记录创建/上次移动的时间 |
| BULLET | direction | DIRECTION | 方向 |
| color | COLORREF | 颜色 |
| REMAIN | direction | DIRECTION | 方向 |
| color | COLORREF | 颜色 |
| BOMB | image | IMAGE\* | 图像 |

|  |  |  |
| --- | --- | --- |
| **类名** | **函数原型** | **详细说明（主要功能）** |
| ATTACK | virtual void attack() = 0; | 进行攻击 |
| bool build(int \_x, int \_y, int \_length，DIRECTION \_direction, CAMP \_camp, int vertical\_length, int vertical\_height, int \_ATK); | 创建攻击物，\_x,\_y-初始坐标，\_direction-方向，\_length-使用者的边长，\_camp-使用者阵营， vertical\_lengt， vertival\_width-该攻击物以垂直方向创建时的长度和高度，\_ATK-攻击力，返回值：创建成功-1，失败-0 |
| virtual void draw() const= 0; | 绘制图像 |
| BULLET | bool build(int \_x, int \_y, int \_length, DIRECTION \_direction, CAMP \_camp, int vertical\_length, int vertical\_height, int \_ATK, COLORREF \_color); | 创建,\_color-子弹颜色,其它同ATTACK::build() |
| REMAIN | bool build(int x, int y, int length, DIRECTION direction, CAMP camp, int vertical\_length, int vertical\_height, int ATK, COLORREF color); | 创建,\_color-静止攻击物颜色，其它同ATTACK::build() |
| BOMB | bool build(int \_x, int \_y, int \_length, DIRECTION \_direction, CAMP \_camp, int \_ATK); | 创建,同ATTACK::build() |

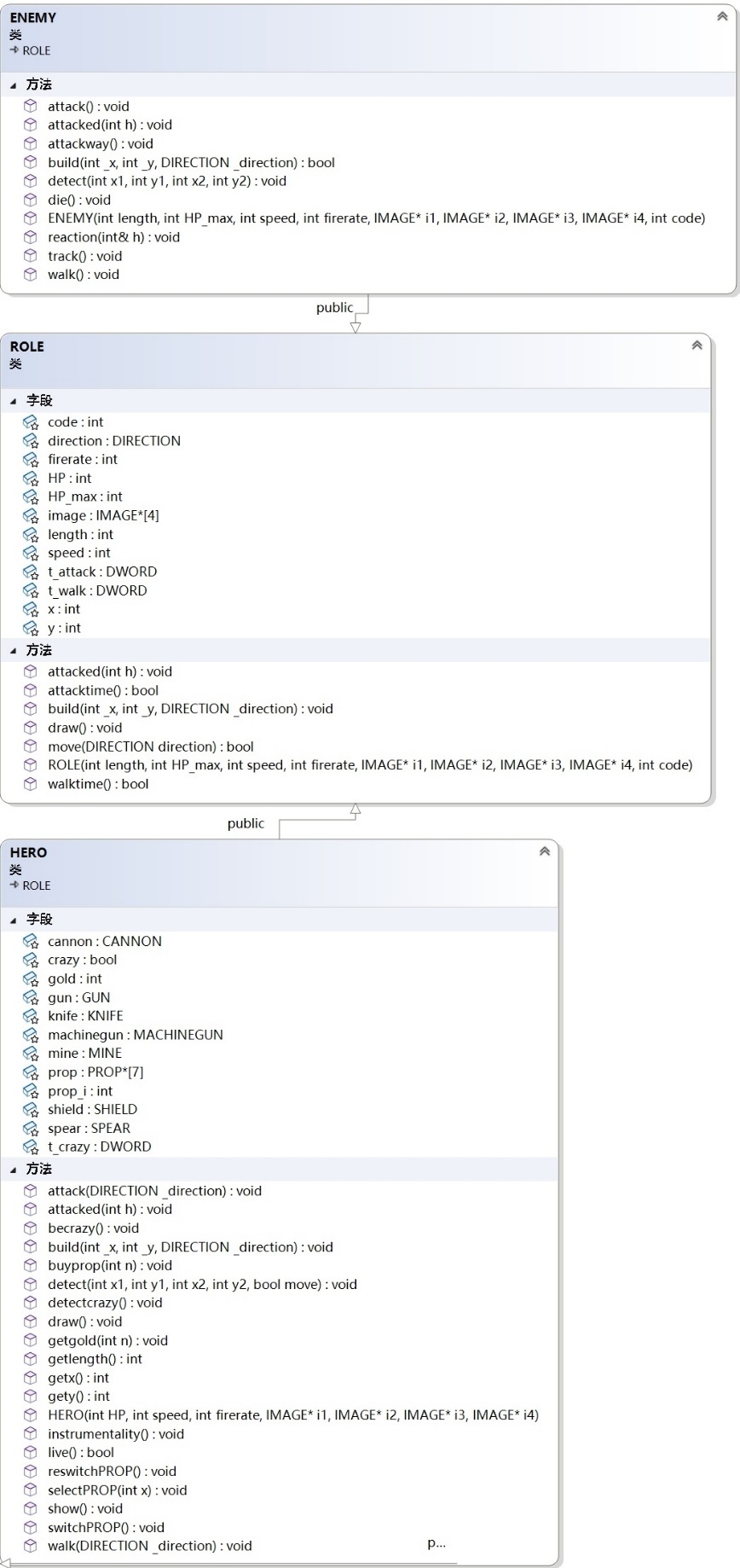
* + 1. **PROP（道具）**

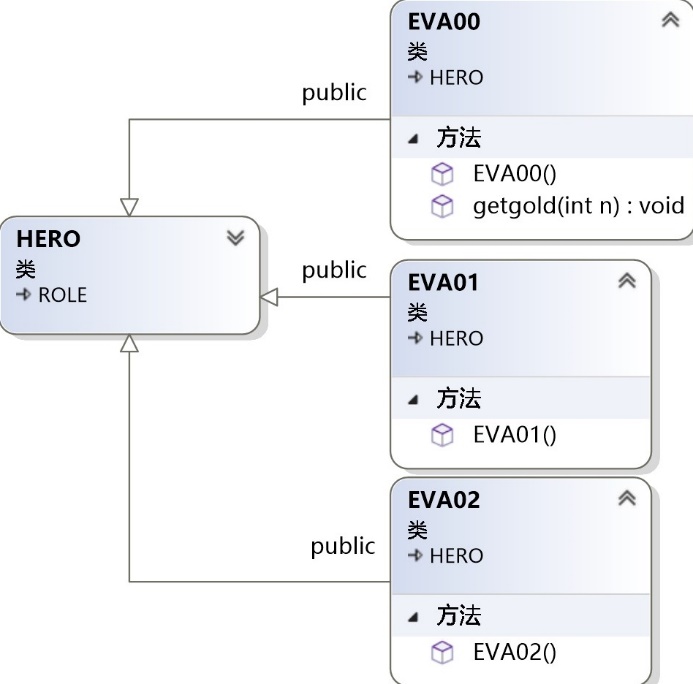


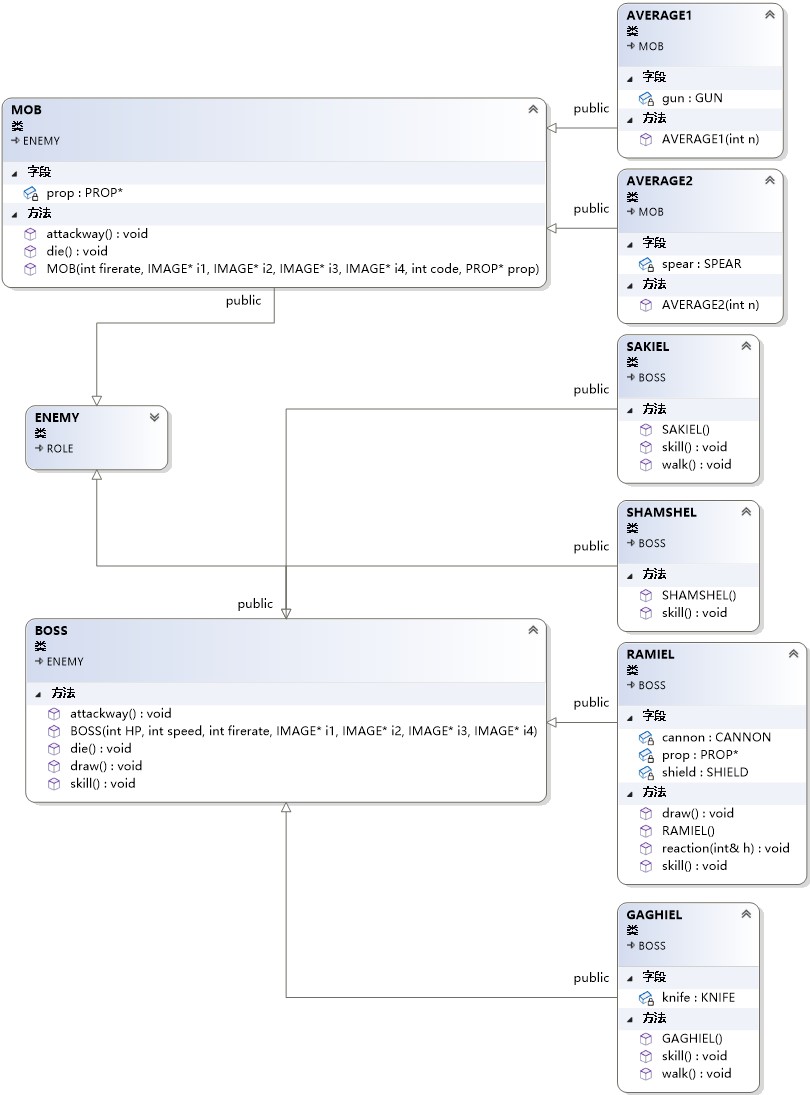
|  |  |  |  |
| --- | --- | --- | --- |
| **类名** | **数据成员名** | **数据类型** | **含义(用途)** |
| PROP | camp | CAMP | 道具使用者所属阵营 |
| length | int | 边长 |
| image | IMAGE\* | 图像 |
| KNIFE | ATK | int | 攻击力 |
| PICKPROP | num | int | 数量 |

|  |  |  |
| --- | --- | --- |
| **类名** | **函数原型** | **详细说明（主要功能）** |
| PROP | virtual void show() const; | 展示道具信息 |
| virtual void use(int x, int y, DIRECTION direction) = 0; | 使用道具，x,y-使用的位置，direction-使用的方向 |
| KNIFE | void changeatk(double n); | 修改小刀的ATK为原来的n倍 |
| PICKPROP | void innum(int n); | 数量增加,n-增加的数量 |
| virtual void skill(int x, int y, DIRECTION direction)const=0; | 每个可拾取道具的特殊能力，x,y-使用者坐标，direction-使用的方向 |
| SHIELD | void defend(int& h); | 防御，吸收伤害，h-伤害 |

* + 1. **ROLE（角色）**

****

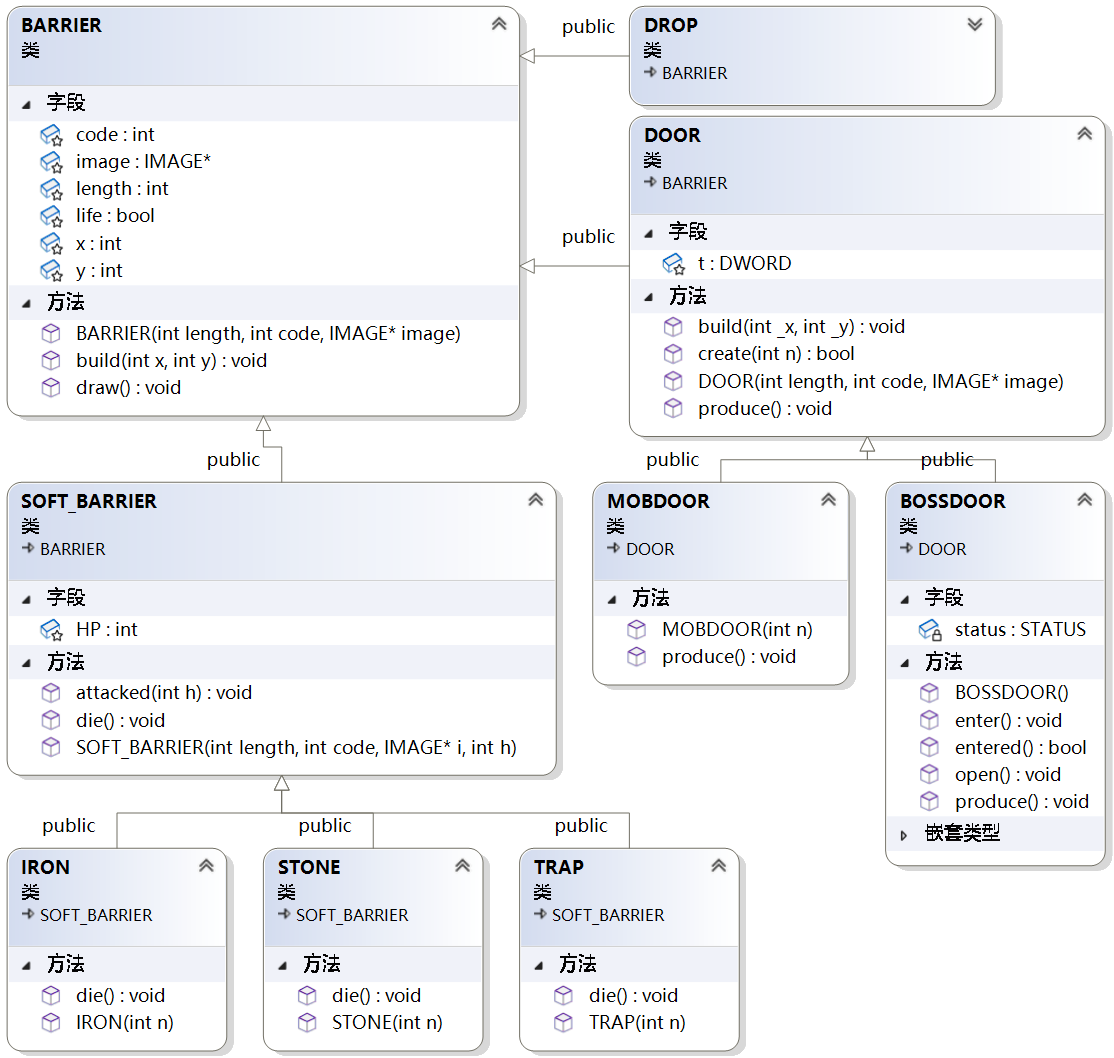


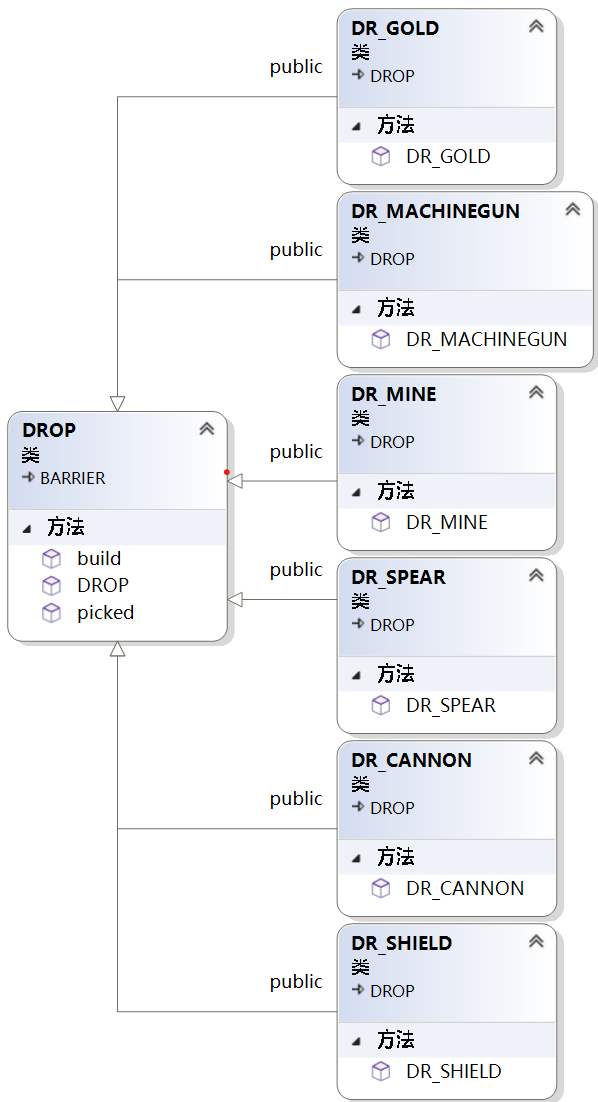


|  |  |  |  |
| --- | --- | --- | --- |
| **类名** | **数据成员名** | **数据类型** | **含义(用途)** |
| ROLE | x | int | 位置横坐标 |
| y | int | 位置纵坐标 |
| HP\_max | int | 生命值上限 |
| HP | int | 生命值 |
| speed | int | 行走速度 |
| firerate | int | 攻击速度 |
| direction | DIRECTION | 方向 |
| length | int | 边长 |
| Image[4] | IMAGE\* | 分别处于四个方向的角色图像 |
| 代码 | int | 代码 |
| t\_walk | DEWORD | 上一次行走的时间 |
| t\_attack | DEWORD | 上一次攻击的时间 |
| HERO | gun | GUN | 玩家的道具 |
| knife | KNIFE |
| machinegun | MACHINEGUN |
| mine | MINE |
| spear | SPEAR |
| cannon | CANNON |
| shield | SHIELD |
| prop[7] | PROP\* | 道具的指针的数组 |
| prop\_i | int | prop[7]的下标，表示当前道具在数组第几个 |
| gold | int | 金币数量 |
| crazy | bool | 是否处于“暴走” |
| t\_crazy | DWORD | 进入“暴走”的时间 |
| MOB | prop | PROP\* | 道具 |
| AVERAGE1 | gun | GUN |
| AVERAGE2 | spear | SPEAR |
| GAGHIEL | knife | KNIFE |
| RAMIEL | cannon | CANNON |
| shield | SHIELD |
| prop | PROP\* |

|  |  |  |
| --- | --- | --- |
| **类名** | **函数原型** | **详细说明（主要功能）** |
| ROLE | void build(int \_x, int \_y, DIRECTION \_direction); | 创建角色，\_x，\_y-初始坐标\_direction-初始方向 |
| bool attacktime(); | 是否可以攻击,1-可以，0-不可以 |
| bool walktime(); | 是否可以行走，1-可以，0-不可以 |
| bool move(DIRECTION direction); | 按方向移动1像素,direction-方向，返回值：1-移动成功，2-移动失败 |
| void draw()const; | 绘制角色图像 |
| virtual void attacked(int h) = 0; | 受攻击，h-伤害 |
| HERO | int getx()const; | 返回x |
| int gety()const; | 返回y |
| int getlength() const; | 返回length |
| bool live()const; | 返回是否活着，1-活着，0-已死亡 |
| void walk(DIRECTION \_direction); | 按方向行走，\_direction-方向 |
| void attack(DIRECTION \_direction); | 按方向攻击, \_direction-方向 |
| void selectPROP(int x); | 选择第x个道具 |
| void switchPROP(); | 按顺序切换道具 |
| void reswitchPROP(); | 按反顺序切换道具 |
| void buyprop(int n); | 购买道具，n-道具的prop\_s代码 |
| void detect(int x1, int y1, int x2, int y2, bool move); | 检测矩形区域内是否有特殊物体，并进行对应操作，x1,y1-矩形左上角坐标，x2,y2-矩形右下角坐标，move-检测前的移动是否成功 |
| virtual void getgold(int n); | 获得金币,n-金币数 |
| void becrazy(); | 进入"暴走" |
| void detectcrazy(); | 检测是否关闭"暴走" |
| void instrumentality(); | 使用"补完" |
| void show()const; | 展示信息 |
| ENEMY | bool build(int \_x, int \_y, DIRECTION \_direction); | 创建一个敌人，\_x，\_y-初始坐标，\_direction-初始方向，返回值：创建成功-1，失败-0 |
| void track(); | 追踪玩家,朝玩家方向前进 |
| void detect(int x1, int y1, int x2, int y2) const; | 检测矩形区域内是否有特殊物体，并进行对应操作，x1,y1-矩形左上角坐标，x2,y2-矩形右下角坐标 |
| void attack(); | 自动攻击 |
| virtual void walk(); | 行走 |
| virtual void reaction(int& h); | 受攻击时的反应，h-伤害 |
| virtual void attackway()=0; | 攻击方法 |
| virtual void die()const = 0; | 死亡奖励 |
| BOSS | virtual void skill()=0; | 专属技能 |

* + 1. **BARRIER（障碍物）**

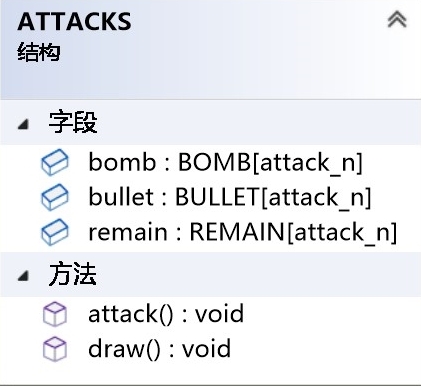
****

****

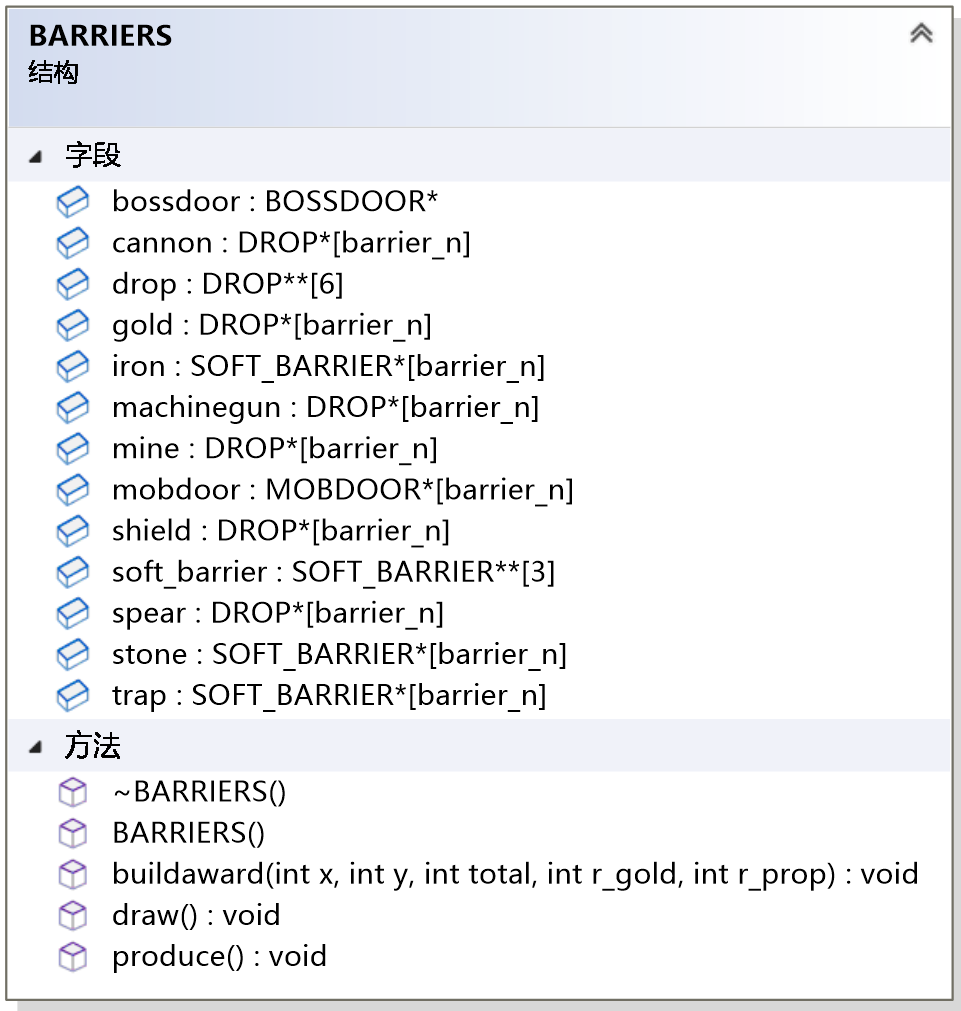
|  |  |  |  |
| --- | --- | --- | --- |
| **类名** | **数据成员名** | **数据类型** | **含义(用途)** |
| BARRIER | x | int | 位置横坐标 |
| y | int | 位置纵坐标 |
| length | int | 边长 |
| Image[4] | IMAGE\* | 分别处于四个方向的角色图像 |
| 代码 | int | 代码 |
| life | bool | 是否已被创建 |
| DOOR | t | DWORD | 被创建的时间 |
| SOFT\_BARRIER | HP | int | 生命值 |
| BOSSDOOR | status | enum class STATUS{produce, shut,open,entered } | 状态,分为生产状态，关闭状态,开启状态,已进入状态 |

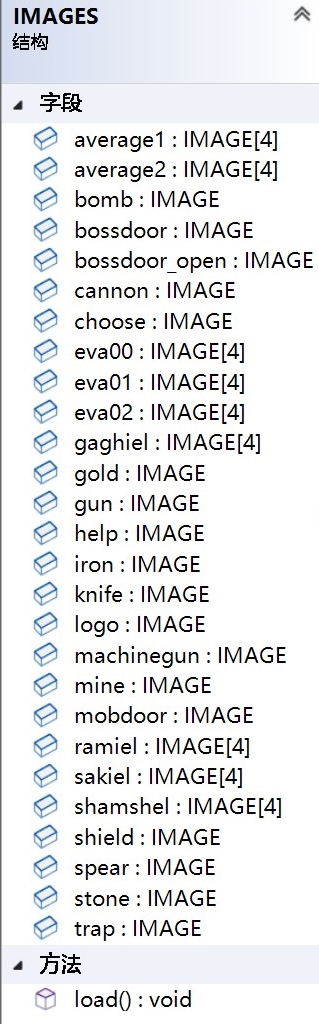
|  |  |  |
| --- | --- | --- |
| **类名** | **函数原型** | **详细说明（主要功能）** |
| BARRIER | void build(int x, int y); | 创建障碍物，x，y-初始坐标 |
| void draw()const; | 绘制 |
| DOOR | bool create(int n); | 生成敌人，n-敌人的代码，返回值：成功生成-1，失败-0 |
| virtual void produce() = 0; | 创建敌人 |
| SOFT\_BARRIER | void attacked(int h); | 受攻击，h-伤害 |
| virtual void die()const = 0; | 死亡奖励 |
| DROP | bool build(int x, int y); | 创建一个掉落物，x,y-初始坐标，返回值：创建成功-1，失败-0 |
| void picked(); | 被拾取 |
| BOSSDOOR | void open(); | 打开出口 |
| void enter(); | 进入出口 |
| bool entered()const; | 返回是否进入了出口，返回值：是-1，否-0 |

* 1. **封装**





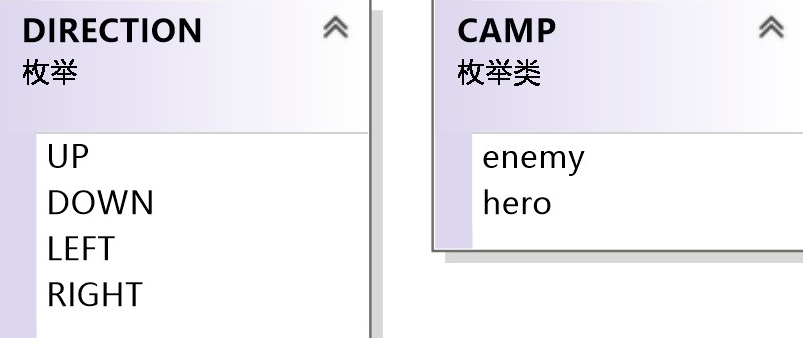
****



|  |  |  |  |
| --- | --- | --- | --- |
| **结构体名** | **数据成员名** | **数据类型** | **含义（用途）** |
| ATTACKS | bomb[attack\_n] | BOMB | 炸弹的数组 |
| bullet[attack\_n] | BULLET | 子弹的数组 |
| remain[attack\_n] | REMAIN | 静止攻击物的数组 |
| ENEMYS | boss | BOSS\* | BOSS的指针 |
| average1[mob\_n] | MOB\* | 量产机1号的指针数组 |
| average2[mob\_n] | 量产机2号的指针数组 |
| mob[2] | MOB\*\* | 小怪派生类指针数组名的数组 |
| BARRIERS | stone[barrier\_n] | SOFT\_BARRIER\* | 石头的指针数组 |
| iron[barrier\_n] | 铁类的指针数组 |
| trap[barrier\_n] | 陷阱的指针数组 |
| mobdoor[barrier\_n] | MOBDOOR\* | 小怪传送门的指针数组 |
| bossdoor | BOSSDOOR\* | BOSS传送门的指针 |
| soft\_barrier[3] | SOFT\_BARRIER\*\* | 可摧毁障碍物派生类指针数组名的数组 |
| gold[barrier\_n] | DROP\* | 金币的指针数组 |
| machinegun[barrier\_n] | 机枪的指针数组 |
| spear[barrier\_n] | 长枪的指针数组 |
| mine[barrier\_n] | 地雷的指针数组 |
| cannon[barrier\_n] | 阳电子炮的指针数组 |
| shield[barrier\_n] | 盾牌的指针数组 |
| drop[6] | DROP\*\* | 掉落物指针数组名的数组 |
| IMAGES | logo | IMAGE | 开始界面图片 |
| help | 帮助界面图片 |
| eva00[4] | 零号机图片 |
| eva01[4] | 初号机图片 |
| eva02[4] | 二号机图片 |
| average1[4] | 量产机1号图片 |
| average2[4] | 量产机2号图片 |
| sakiel[4] | 水天使图片 |
| shamshel[4] | 昼天使图片 |
| ramiel[4] | 雷天使图片 |
| gaghiel[4] | 鱼天使图片 |
| stone | 石头图片 |
| trap | 陷阱图片 |
| iron | 铁图片 |
| mobdoor | 小怪传送门图片 |
| bossdoor | BOSS传送门图片 |
| bossdoor\_open | BOSS传送门（开启状态）图片 |
| gun | 步枪图片 |
| knife | 小刀图片 |
| machinegun | 机枪图片 |
| mine | 地雷图片 |
| spear | 长枪图片 |
| cannon | 阳电子炮图片 |
| shield | 盾牌图片 |
| gold | 金币图片 |
| bomb | 炸弹图片 |

|  |  |  |
| --- | --- | --- |
| **类名** | **函数原型** | **详细说明（主要功能）** |
| ATTACKS | void attack(); | 攻击 |
| void draw()const; | 绘制 |
| ENEMYS | void walk(); | 行走 |
| void attack(); | 攻击 |
| void draw()const; | 绘制 |
| BARRIERS | void draw()const; | 绘制 |
| void produce(); | 生成敌人 |
| void buildaward(int x, int y, int total, int r\_gold, int r\_prop); | 生成一个掉落物，x,y-位置，r\_prop/total-每个道具的概率，r\_gold/toal-金币概率 |
| IMAGES | void load(); | 加载图像 |

* 1. **枚举**



* 1. **全局变量、函数**

|  |  |  |
| --- | --- | --- |
| **数据成员名** | **数据类型** | **含义(用途)** |
| hero | HERO\* | 玩家指针 |
| enemys | ENEMYS\* | 敌人结构体指针 |
| barriers | BARRIERS\* | 障碍物结构体指针 |
| attacks | ATTACKS\* | 攻击物结构体指针 |
| images | IMAGES\* | 图像结构体指针 |
| map[HEIGHT][LENGTH] | int | 地图 |
| score | int | 游戏分数，每秒减1分，每杀死一个敌人奖励分数 |

|  |  |
| --- | --- |
| **函数原型** | **详细说明（主要功能）** |
| void init(); | 初始化 |
| void start(); | 开始界面 |
| void set(); | 设置界面 |
| void help(); | 帮助界面 |
| void choose(); | 角色选择界面 |
| bool play(); | 游戏界面，返回值：胜利-1，失败-0 |
| bool round(int n); | 游戏关卡，n-关卡数，返回值：胜利-1，失败-0 |
| void init\_map(); | 加载地图 |
| void shop(); | 商店界面 |
| void end(bool result); | 结束界面，result-游戏结果 |
| void start\_draw(); | 绘制开始界面 |
| void set\_draw(); | 绘制设置界面 |
| void choose\_draw(); | 绘制角色选择界面 |
| void round\_draw(); | 绘制游戏画面 |
| void shop\_draw(); | 绘制商店界面 |
| void end\_draw(); | 绘制结束界面 |
| void clear(); | 清除键盘输入缓冲区,防止错误的\_getch() |
| void setmap(int x, int y, int length, int code); | 设置正方形区域的代码，x,y-正方形左上角坐标，length-正方形边长,code-代码 |
| bool empty(int x1, int y1, int x2, int y2); | 检测矩形区域是否为空，x1,y1-矩形左上角坐标，x2,y2-矩形右下角坐标，返回值：有物体-0，区域为空-1 |
| bool empty(int x, int y, int length); | 检测正方形区域是否为空，x,y-正方形左上角坐标，length-正方形边长，返回值：有物体-0，区域为空-1 |
| bool harm(int x1, int y1, int x2, int y2, CAMP camp, int ATK); | 检测矩形区域内能受伤的物体,并造成伤害，x1,y1-矩形左上角坐标，x2,y2-矩形右下角坐标，camp-伤害来源所属阵营，ATK-攻击力，返回值：1-造成了伤害，0-没造成伤害 |
| void harmnear(int x, int y, int length, int range, CAMP camp, int ATK); | 分别对四个方向以length为长，range为宽的矩形区域造成伤害，x,y-伤害来源坐标，length-伤害来源边长，range-伤害范围，camp-伤害来源所属阵营，ATK-攻击力 |
| void shownum(int x, int y, int n); | 在指定位置显示整数变量，x,y-整数左上角坐标，n-整数数值 |

* 1. **常量**

|  |  |  |  |
| --- | --- | --- | --- |
| **数据成员名** | **数据类型** | **含义(用途)** | |
| LENGTH | constexpr int | 尺寸 | 画布长 |
| HEIGHT | constexpr int | 画布高 |
| small\_length | constexpr int | 边长（小） |
| midium\_length | constexpr int | 边长（中） |
| large\_length | constexpr int | 边长（大） |
| empty\_s | constexpr int | 地图代码 | 空 |
| hero\_s | constexpr int | 玩家 |
| boss\_s | constexpr int | BOSS |
| average1\_s | constexpr int | 量产机1号 |
| average2\_s | constexpr int | 量产机2号 |
| bossdoor\_s | constexpr int | BOSS传送门 |
| stone\_s | constexpr int | 石头 |
| trap\_s | constexpr int | 陷阱 |
| iron\_s | constexpr int | 铁 |
| gold\_s | constexpr int | 金币 |
| mobdoor\_s | constexpr int | 小怪传送门 |
| machinegun\_s | constexpr int | 机枪 |
| mine\_s | constexpr int | N2地雷 |
| spear\_s | constexpr int | 朗基努斯之枪 |
| cannon\_s | constexpr int | 阳电子炮 |
| shield\_s | constexpr int | 盾牌 |
| mob\_n | constexpr int | 每种重复对象的数量  （数组的大小） | 每种小怪的数量 |
| barrier\_n | constexpr int | 每种障碍物的数量 |
| attack\_n | constexpr int | 每种攻击的数量 |
| MAP[3][26][26] | constexpr int | 预先准备好的地图 | |

1. **主要技术难点及实现方案/算法设计**
   1. **循环游戏**

游戏要能够循环进行（重新开始），所以主函数中有一个大的while循环，然后实现每个界面的函数中都有一个while循环，只有完成特定操作，循环才会结束。大循环只能在结束界面中用exit（0）结束，否则会重新开始一轮新游戏。

* 1. **碰撞检测**

行走和攻击时都要知道某个位置是否有对象、有哪个对象，来判断是否可以前进、击中了哪个目标，但如果直接根据对象的坐标来判断，每次检测时都得遍历所有对象，代码就会非常繁琐。

每个有碰撞体积的对象都设置一个唯一的代码(该对象在地图上显示的代号)，而空位置的代码为empty\_s。定义一个二维数组map来记录每个位置的代码。物体移动到哪，就将之前位置的map数值改为empty\_s，现在位置的map数值改为专属代码。其它对象通过检测某区域的map数值，就能知道该区域有哪个物体。

* 1. **对象数组无法多态，基类指针不能调用派生类数组**

基类指针不能用派生类的数组名赋值，因为基类指针使用下标时会以基类对象的大小作为数组元素的大小，来寻找每个元素的首地址，可派生类的数据一般会比基类多，导致寻址错误。但游戏的对象数组较多，一些派生类对象只需要基类指针就能满足调用需求（利用虚函数），如果能实现多态，还可以减少函数的重载（形参统一为基类指针即可）。

所以我选择先定义基类指针的数组，在for循环中为每个指针动态分配一个派生类对象，这样就能用基类指针的数组名来访问基类指针，进而调用每个派生类对象。

* 1. **保证同类的对象各自有唯一的代码**

游戏中有大量同类对象（比如有100个小怪），但每个对象的代码得是唯一的，而且要和它的类以及数组下标相关，才能靠map数值反推出是哪个对象。

首先为每个对象数组的第一个对象定义一个常量（首代码），再定义对象数组的大小，就能确定同类对象的代码所在的区间，也方便保证不同类对象的代码不会重复。然后，在2.2中所说的为指针分配派生类对象的过程中，用指针的下标作为构造函数的实参，将记录代码的成员变量赋值为首代码+数组下标，实现数组下标和代码的对应。

* 1. **在游戏中创建一个对象**

如果要在游戏中途构造一个对象，实现难度大且开销较大。但有些对象需要循环出现，如利用小怪传送门每隔一段时间创建一个小怪。

可以在每轮游戏开始时先构造这个对象，对象有一个成员变量life来记录是否已被创建，并且有个build函数，当life=0（未被创建）时执行创建操作并返回1，life=1（已被创建）时返回0。想创建一个对象时就用for循环遍历对象数组调用build函数，当函数返回1时停止循环。对象死亡时life赋值为0，之后就又可以被创建。

* 1. **实现速度**

为了实现攻击（移动）速度（即每两次攻击（移动）之间有固定的时间间隔），为攻击和移动分别定义一个DWORD类型的变量t，当角色移动时，用GetTickCount函数将开机以来经过的时间（可以当作时钟）赋值给t，然后必须经过固定的时间后（GetTickCount（）-t>时间间隔），角色才能继续移动，接着再给t赋值。

* 1. **全局变量过多，函数分布分散**

游戏中各个类的对象互相关联，一些类需要在源文件调用其它类对象的函数，这就必须先声明被调用的类的具体对象为全局变量，导致全局变量过多。而且各个对象的函数的调用分散在多个文件中，难以查找，可读性和可改性差。

利用结构体封装多个派生类的对象，结构体的指针设置为全局变量，就可以在多个文件里使用结构体指针调用对象。将对象的相似的函数（如绘制函数）集中封装在结构体的函数中，在其它文件里使用结构体的函数。如果新增了派生类，只需要在结构体文件里引用头文件、增加对象和函数就行。

* 1. **怪物自动追踪和攻击玩家**

因为怪物都是从自身的中间发动攻击，所以只要怪物的对称轴与玩家相交，攻击就一定能击中玩家，这就能确定怪物的有效攻击范围。只要玩家在怪物的攻击范围内，怪物就直接朝玩家移动和攻击。如果不在攻击范围内，怪物就根据玩家相对自身的方向沿折线移动，二者相对距离较大的方向为首选方向，另一个方向为备选方向，如果朝首选方向移动失败（比如遇到障碍物），就改为朝备选方向移动。

* 1. **键盘缓存导致错误输入**

键盘输入有缓存区，\_getch函数会从缓存区取值，造成意外的输入。特别是，当一轮游戏结束时，可能之前的一些输入还没有执行，到下一轮游戏开始时就会被自动读取，影响到角色选择和设置。

利用循环while (\_kbhit())\_getch();，能读取完之前的全部键盘输入，来清空缓存区。

* 1. **游戏不流畅**

对于画面闪烁和卡顿，用BeginBatchDraw函数开启双缓冲作图，所有图像先在内存中画好，再显示在画布上，保证图像同时显示且变化更流畅。

在游戏中，利用\_getch函数读取操作键（WASD和方向键）输入会有延时和卡顿。用GetAsyncKeyState函数替代\_getch函数，可以直接判断某个键是否被按下，反应速度更快。