课程设计2-植物大战僵尸报告

- 选题：标准选题（植物大战僵尸）
- 姓名：马兴越
- 学号：171870660
- 院系：现代工程与应用科学学院
- 编写调试平台：Windows 10, Microsoft Visual Studio 2019

提交目录

- `project` 目录，包含完整的Visual Studio工程文件。
- `bin`目录，在Windows 64位编译出的可执行文件。

实现摘要

对照课程设计的要求，此处简单列出实现的内容。这里仅仅列出实现结果，而不具体解释实现方法。

特色之处

- 植物、僵尸、炮弹的公共抽象基类`Placeable`及类层次。
- 僵尸及护具的继承、聚集混合实现方法，以及魔改组合各种护具和僵尸；带护具僵尸的模版类及mixin设计方法。
- 植物、僵尸工厂类，包括抽象基类和模版的实际类。

系统与场地

系统有一时间戳，所以时刻表达都通过**时间戳**，时间表达都以**时钟周期**为单位。。

- 5行8列庭院，自动产生自然光，植物购买（地块选择，种植等），铲子，计分板。
- 按一定逻辑产生僵尸，且有一定的难度递增。
- 提示信息栏，系统时间戳显示

植物

植物具有公共接口，都包含生命值，耗费阳光数量，冷却时间等。具体包括下列的植物。对攻击型的植物，单独有`Seed`类表达植物的“炮弹”。这里简单列出所有实现的植物及其基本参数。

特别指出，在设计中，“攻击力”并非是植物的属性，而是植物的炮弹（`Seed`）的属性。但由于设计中的每一种植物都唯一对应到一种炮弹，方便起见，这里也直接列出。

名称	阳光	冷却时间	生命值	特征周期	攻击力
向日葵	50	5	10	15	-
豌豆射手	100	5	10	4	2
寒冰射手	175	10	10	4	2
双发射手	200	10	12	4, 1	2
西瓜投手	300	20	15	5	5
坚果墙	50	40	60	-	-
地刺	100	20	-	1	1
土豆雷	25	50	10	20	∞
火爆辣椒	300	50	-	2	∞
樱桃炸弹	300	50	-	2	∞

僵尸及生成逻辑

僵尸大致由两类，单独的一只僵尸（用纯粹的继承实现）和僵尸加上护具（继承+聚集实现）。理论上僵尸可以任意魔改组合，比如说铁桶撑杆僵尸，路障摇旗僵尸之类的。

每500个时钟周期产生一大波僵尸，一大波僵尸总是由摇旗僵尸领头，时长80个时钟周期，这段时间产生僵尸的概率会很高。

一般的僵尸有一个产生相对概率，及一个起始时刻，也就是在该时刻指定的时间戳之后，才可能产生这种僵尸，以体现难度递增。

这里列出所有玩家实际看到的僵尸，也就是包括了僵尸和护具的组合。斜体标出名称的僵尸属于这种类型。所有的僵尸的攻击力（每吃一口的杀伤力）都是2。表中的名称和生命值都已经包括了护具，特征周期是指吃两口之间的间隙。

名称	生命值	速度	分值	特征周期	相对概率	起始时刻
僵尸	10	2	1	3	10	50
路障僵尸	20	2	2	3	10	80
垃圾桶僵尸	30	2	3	3	8	100
铁桶僵尸	40	2	4	3	8	120
读报僵尸	16	2, 6	3	3	6	140
门板僵尸	50	2	5	3	6	200
撑杆僵尸	20	2	3	3	5	250
玩偶盒僵尸	20	2	3	3	4	320
橄榄球僵尸	40	3	6	2	6	400
铁桶撑杆僵尸	50	2	6	3	6	800
摇旗僵尸	15	3	2	3	-	-

架构设计

整套系统设计可以大致分为以下几个大块：

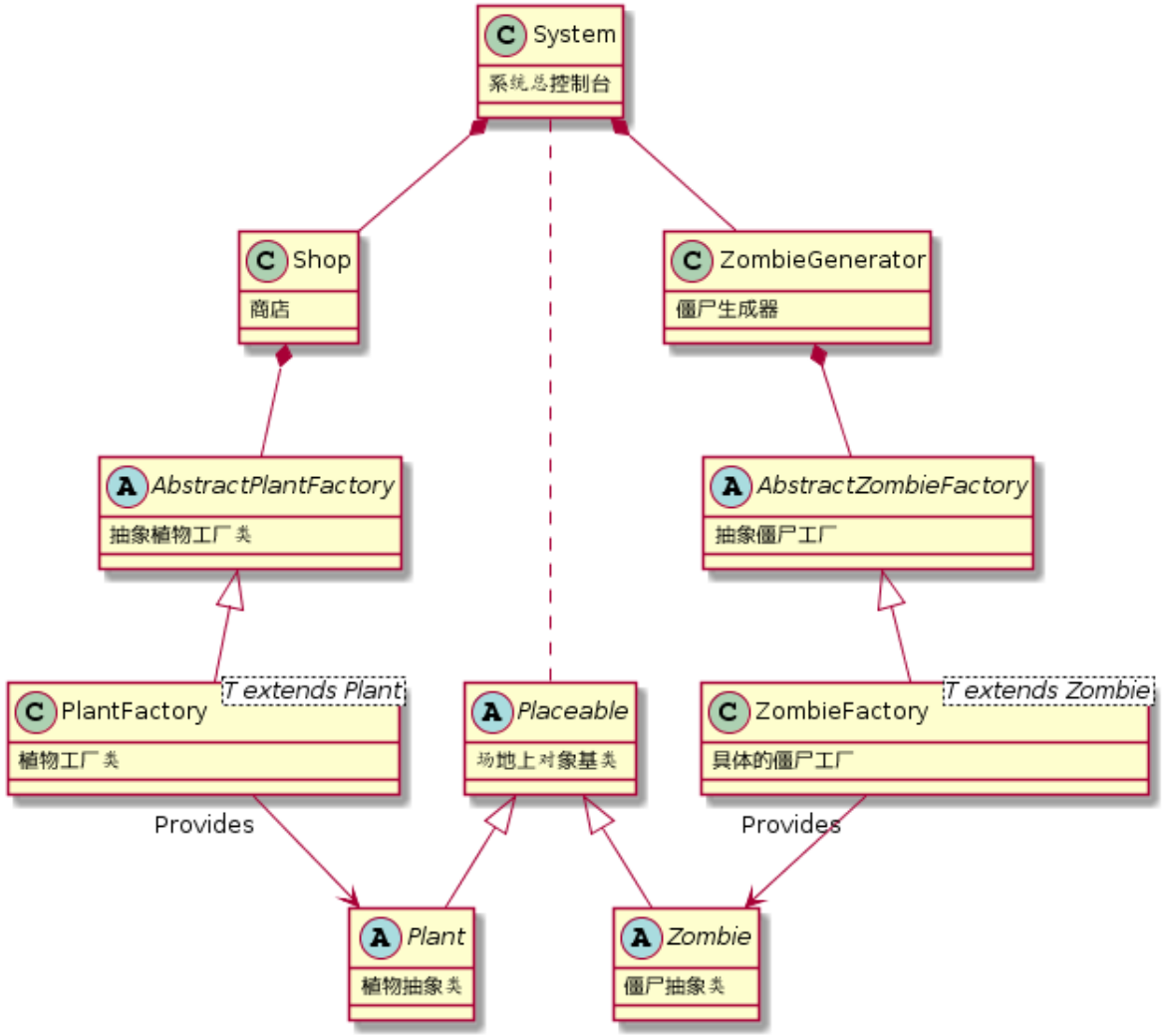
- 底层部分，封装基本的控制台交互，包括`Terminal`一个类。
- 场地部分，即庭院和地块，主要包括`Yard`和`Block`两个类，后者强合成于前者中。另有`Position`类，场地上对象和场地之间的桥梁。
- 场地上对象部分。这部分所有的类具有公共的抽象基类`Placeable`，并有以下三个抽象子类。所有的实体类都有这三个类派生而来。
  - 植物部分，抽象基类为`Plant`。
  - 植物炮弹（即种子）部分，抽象基类为`Seed`。植物和炮弹的主要实现类的头文件和实现文件放在`plants`目录下。
  - 僵尸部分，抽象基类为`Zombie`。此外还包括僵尸的护具，抽象基类为`AbstractProtector`，这两部分文件放在`zombies`目录下。
- 控制与交互部分。这部分包括主系统`System`，商店`Shop`，僵尸生成器`ZombieGenerator`以及关联植物/僵尸与生成部分的工厂类。

由于整套系统设计的类比较多，每个类的逻辑实际上都比较简单，但具有较为复杂的关联关系。为了尽可能突出重点，下面的叙述中，针对上面的每个部分（除底层部分外），给出一张不大规范的`UML`类图来反映类之间的关系，而另外用文字或者代码形式描述类定义。

控制与交互部分

为了从全局视角，尽量高屋建瓴地阐述，这里最先给出控制与交互部分的设计。

`UML`类图



`System`类

这是系统的中枢。它包含庭院类`Yard`，商店类`Shop`，僵尸生成器`ZombieGenerator`的值，并直接实现以下功能：

- 记录阳光和积分。
- 周期性自动产生阳光。
- 管理所有的`Placeable`对象的指针。
- 主事件循环（函数`mainLoop`）。通过`Sleep()`控制时钟周期，读取键盘输入，并调用购物、铲除植物、退出等函数。

植物工厂

首先给出抽象类的定义中有意义的部分。设计抽象的植物工厂，主要是为了在`Shop`中产生植物时，有一个统一的接口，方便构造数组。后面将看到，`Shop`中与植物工厂的交互都通过这个抽象类完成。

植物所有的[只与购买前后有关的性质]都保存在`PlantFactory`中，这包括冷却时间、耗费阳光数。

```
class AbstractPlantFactory
{
protected:
    const int coldPeriod, cost; //冷却时间和耗费阳光。对于一种特定的植物来说，它们是常量。
    const std::string name;
public:
    AbstractPlantFactory(System& sys, int cold, int cost,const std::string& n);
    inline bool available()const;  //是否装载完成
    inline int getRate()const;  //装填完成比例，用于在商店界面显示
    virtual Plant* newInstance() = 0;  //产生植物对象。
};
```

实际的植物工厂是通过继承抽象类的模版类实现的。通过这种方法，无需为每一种植物单独设计工厂类，在增加一种植物时，只需要在`Shop`中增加一行代码即可。继承类的形式很简洁：

```
template <typename T>
class PlantFactory :
    public AbstractPlantFactory
{
public:
    PlantFactory(System& sys,int coldPeriod,int cst, const std::string& name);
    virtual T* newInstance()override;
};
```

`Shop`类

负责完成用户购物时候的交互部分，所有的植物添加都通过这里完成；同时负责更新庭院以下部分的界面。这里给出一些部分的类定义。

```
class Shop
{
    AbstractPlantFactory* factories[N];
public:
    Shop(System& sys);
    void buy();//涉及交互，负责向yard添加植物
    void updateUI();  //每个时钟周期更新界面
private:
    void initUI();
};
```

其中关于`PlantFactory`的初始化形式大概是：

```
Shop::Shop(System& sys) :
    factories{
        new PlantFactory<Sunflower>(sys,5,50,"向日葵"),
        new PlantFactory<PeaShooter>(sys,5,100,"豌豆射手"),
        ...
    }
...
```

僵尸工厂

僵尸工厂的设计思路和继承结构和植物工厂完全类似，这里不再赘述。只是与构造有关的具体数据不大相同。僵尸工厂类保存的数据是僵尸产生的相对概率，和该种僵尸可能开始产生的时间点。

`ZombieGenerator`类

与`Shop`负责产生植物实例类似，这个类负责产生僵尸的实例。部分的类定义为

```
class ZombieGenerator
{
    static const int GroupInterval = 500, GroupLength = 80; //“一大波僵尸”的周期和时长
    AbstractZombieFactory* factories[N];
    AbstractZombieFactory* directFactory;  //摇旗僵尸
public:
    void generate();  //接口方法，由System直接调用
private:
    double rate()const;  //返回当前时钟周期产生僵尸的概率
    int truns()const;
};
```

僵尸产生的方法是：每个时钟周期有`turns()`次机会产生僵尸，

> 可以理解为，每个周期掷`turns()`次骰子。在非“一大波”的状态，`turns()`总是1；在一大波的状态，第`n`轮中`turn()`返回值为`n`，最大为`5`。

每一次机会中有`rate()`的概率产生僵尸，

> 在非“一大波”的状态下，`rate()`一般返回0.5；特别的，在游戏开始后第一大波僵尸到达前，这个概率从`0.1`开始线性地增加到`0.5`。在“一大波”的状态下，`rate()`总是返回0.90. 这就是说，第5轮及之后的一大波僵尸中，每一个时钟周期将会期望产生4.5只僵尸。

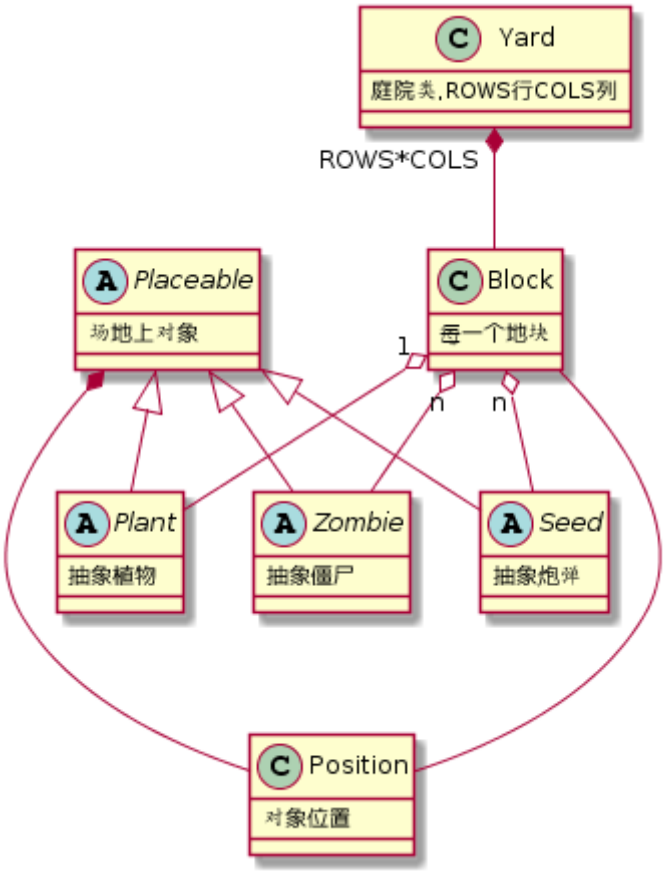
如果确定产生僵尸，则产生各种当前可能产生的僵尸，按照相对概率产生。每一轮的僵尸生成之间是独立的。

概括地说，游戏难度的渐进性体现在：

- 每一种僵尸只能在一定的时间点之后才能产生。
- 第一轮之前，产生僵尸的概率是随时间线性增加的。
- 在前5轮中，每一大波的僵尸的数量是递增的。

场地部分

`UML`类图



`Block`类

每个`Block`对象指庭院中的一个地块，它由控制台中一定的行列数构成。

`Block`中分类别保存了该地块上植物，所有当前在该地块上的僵尸和炮弹的指针。当僵尸（炮弹）走入（飞入）地块时，将其指针加入；离开时取出。每个时钟周期，`Block`更新地块上要显示的内容。

`Yard`类

庭院类，可以理解成`ROWS*COLS`个`Block`的组合，持有他们的值类型对象。负责更新界面。每个时钟周期刷新一次界面。

`Position`类

位置类，描述某一个特定的场地上对象当前的位置。对于每一个场地上对象（`Placeable`对象），我们约定其位置由行、列表示，其中行只精确到庭院上抽象的一行，而列精确到具体的字符位置（或理解为控制台的“像素”，程序中用`colpix`区分）。重要的接口方法是

```
Block* Position::target();
```

指出当前位置对应庭院上的地块。

场地上对象部分

`Placeable`基类

前面已经提到，所有的植物、僵尸、炮弹，都可以理解为“放”在场地上的，它们都具有一个位置，都要放置和移出，每个时钟周期都要更新一次。根据这些特点，抽象出基类`Placeable`。主要的定义如下：

```
class Placeable
{
protected:
    Position position;
public:
    static int timestamp;  //系统的时间戳是由这里维护的
    virtual void update() = 0;  //每个周期的更新
    virtual void place() = 0;
    virtual void remove() = 0;
    Block* getBlock();
    void setPosition(int row, int colpix);
};
```

在实际编程中，由于`Block`在处理植物、僵尸、炮弹时，使用了不同的逻辑分别管理，所以`place()` `remove()`方法都是分别实现的。这个抽象的基类除了逻辑上的意义外，目前只有一处直接的作用，就是在`System`中保存了所有`Placeable`对象的指针，在每个时钟周期，通过这里调用`update()`函数完成更新。