

课程设计二

植物大战僵尸（控制台版）

Plants Vs. Zombies

时间：2019. 10. 22 - 2019. 11. 17

姓名：唐金麟 17 级

联系方式：TangJinlin@smail.nju.edu.cn

目录

植物大战僵尸（控制台版） - PVZ.....3

一、概述3

 主要内容：3

 已实现的目标：3

二、主要类的设计3

 1) 依次介绍各个主要类的设计：3

 2) 综上所述，各个类之间的关系大致如下图所示： 11

 3) 12 种植物的设计细节 11

 4) 11 种僵尸的设计细节 14

三、程序的功能特点和运行操作方法 16

四、代码实现中值得一提的地方 18

 1) 画面刷新机制 18

 2) 子弹飞行 19

 3) 复用基类代码 19

植物大战僵尸（控制台版） - PVZ

一、概述

主要内容：

以塔防游戏 [植物大战僵尸](#) 为基础参考，实现了运行在 Windows 控制台下的一个游戏。玩家目标是合理利用阳光、植物布局，守住一波一波的僵尸的攻击，击杀僵尸，获得更高的分数。游戏中，任何一只僵尸到达地图的左边界，则游戏结束。玩家可使用"1-9"数字键以及"a-c"键选择要种植植物、方向键控制格子选择框移动、"x"键铲除植物、回车键确定、Esc键返回，空格键暂停（游戏按键均不区分大小写）。

已实现的目标：

- 12 种植物：向日葵、豌豆射手、窝瓜、樱桃炸弹、坚果墙、寒冰射手、双发射手、大蒜、火爆辣椒、土豆地雷、地刺、高坚果墙。
- 11 种僵尸：普通僵尸、摇旗僵尸、路障僵尸、铁桶僵尸、橄榄球僵尸、铁门僵尸、读报僵尸、撑杆僵尸、小丑僵尸、舞王僵尸、伴舞僵尸。
- 较好的 UI 效果：下方显示状态信息、帮助信息（随状态不同而展示不同的帮助信息）；爆炸、小丑、舞王等动作有一定的显示效果；植物名字分三类色块展示……
- 僵尸一波一波地刷新，且随着玩家得分增长，每次刷出的僵尸数量逐渐增加，难度渐进。
- 计分规则：击杀不同的僵尸可获得不同的分数；且游戏持续时间，也会使得分数增加（生存得分），1 秒/分。

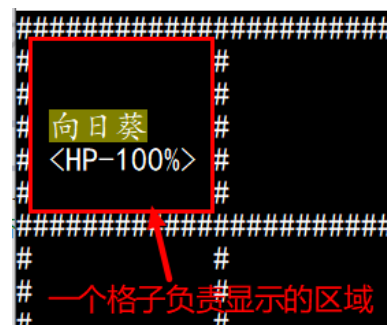
二、主要类的设计

1) 依次介绍各个主要类的设计：

a) Grid 类

这里的一个 Grid 类对象，是游戏草地中的一个“格子”，游戏地图中的一个小单元。一个格子中，可有的元素为：1 或者 0 个植物、大于等于 0 个僵尸。这里 Grid 类最主要的作用就是如何展示本格子内的元素，以及对外提供对本格元素的一些操作功能，比如设置是否选中、增加/删除植物、增加/移除一个僵尸等。

部分数据成员及成员函数展示：



```

10  class Grid {
11      //格子左上角的坐标, 用于绘制时加上的偏移量
12      int dx, dy;
13      //此格子中的植物
14      Plant* plant;
15      //格子中的僵尸
16      vector<Zombie*> zombies;
17
18      /*其他辅助信息*/
19      //是否选中
20      bool selected;
21      //用于标记是否需要刷新显示
22      bool flag_refresh;
23      void setRefresh() { flag_refresh = true; }
24      //用于标记本格子是否处于爆炸特效 (樱桃炸弹等)
25      bool bomb_flag;
26
27      //根据格子的坐标来设定dx,dy
28      void setXY(int x, int y);
29      //绘制格子中的显示内容
30      void paint();
31      //设定植物
32      bool setPlant(Plant* iPlant);
33      //清空植物
34      void delPlant() { delete plant; plant = nullptr; flag_refresh = true; clearBombFlag(); }
35      //加入一个僵尸
36      void addZombie(Zombie* iZombie);
37      //清除一个特定的僵尸
38      void delZombie(Zombie* iZombie);
39      //格子内的所有僵尸扣除attack的血量
40      void hitZombies(int attack);
41
42

```

这里根据格子内的内容, 有多种显示模式:

```

#####
#
# (蓄力12%) #
# 土豆地雷 #
# <HP-100%> #
#
#####

```

①只包含植物: (包含植物的一些额外信息)

```

#####
#
# 坚果墙 #
# <HP-98%> #
# 僵尸×1 #
#
#####

```

②包含植物和僵尸:

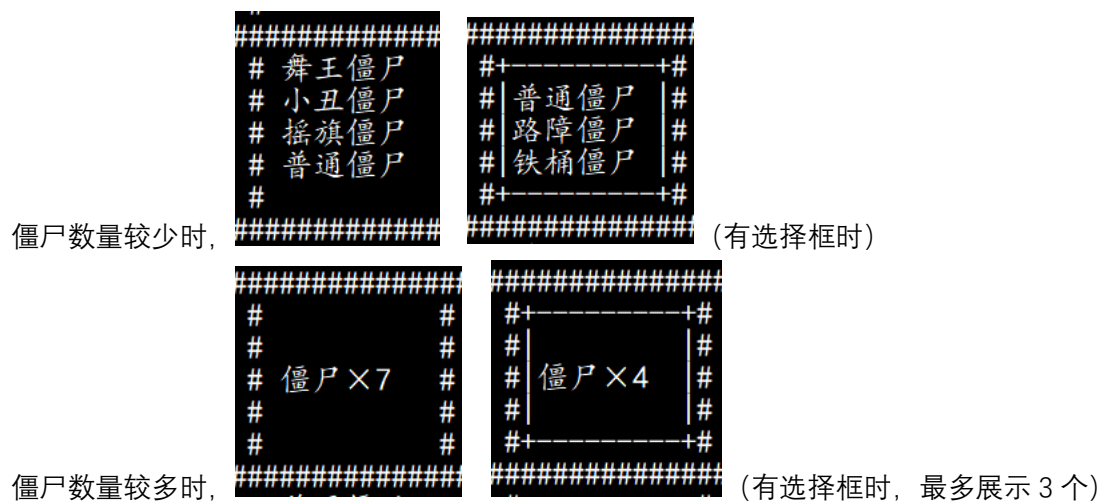
③只包含僵尸时:

```

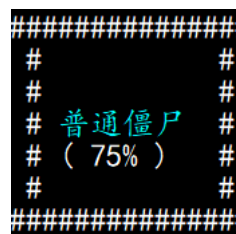
#####
#
# (有撑杆) #
# 撑杆僵尸 #
# ( 100% ) #
#
#####

```

一个僵尸时, (包含僵尸的特殊信息: 有无撑杆/报纸/铁门等等)



此外，还有一些特殊效果的显示，比如：
寒冰射手击中后的僵尸（移动/攻击速度减慢状态）：



读纸僵尸报纸被打烂后的红眼状态（速度变得很快）：

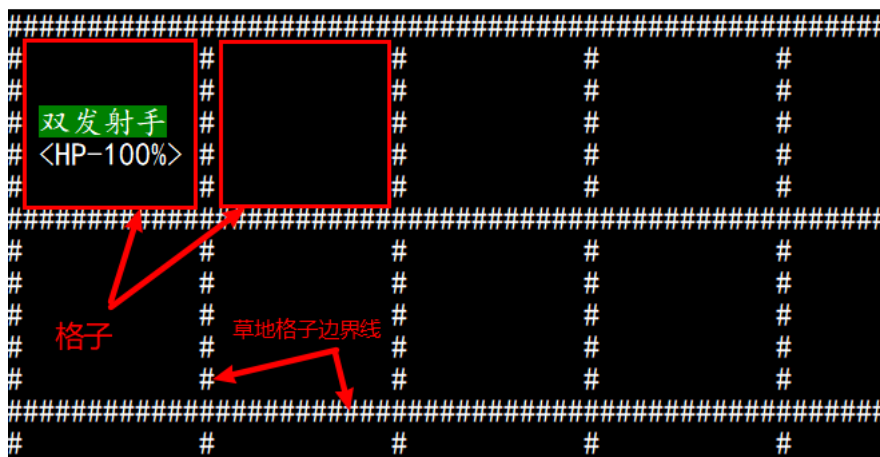


格子的“爆炸”效果（窝瓜、土豆雷、樱桃炸弹、火爆辣椒、小丑僵尸等等）：



b) Map 类

从整个地图的布局来看，地图就是由二维的多个“草地格子”以及格子间的边界线组成。如下图：



所以，这里就自然而然的就有了：Map 类。用于表示整个的游戏地图，包含一个 Grid 类对象的二维数组。主要是提供一些地图层面上的功能，比如地图初始化，刷新显示，负责绘制地图（包括格子和边界线）等等。

部分数据成员及成员函数展示：

```

73  class Map {
74      Grid grid[GRID_NUM_X+1][GRID_NUM_Y];
75  public:
76      //地图的初始化
77      void init();
78      //网格线 (草地块的边界) 的绘制
79      void paintGridLine();
80      //遍历网格: 处理植物、僵尸
81      bool travGrid(Game &game);
82      //种植某类植物
83      bool setPlant(int ix, int iy, int type);
84      //遍历所有格子, 检查是否需要刷新显示
85      void refresh();
86      //(x,y)格子开启/关闭爆炸特效
87      void setBombFlag(int x, int y);
88      void clearBombFlag(int x, int y);
89  
```

c) PlantCard 类

游戏涉及到植物的购买，而植物的购买还涉及到价格、冷却时间等信息，以及这些信息在对应位置的展示，所以这里商店中展示出的可购买植物的信息分装成一个类：PlantCard 类（意为“植物卡片”）。负责某个植物卡片的信息的设定、展示输出。

部分数据成员及成员函数展示：

<pre> 10 //用于商店展示的植物卡牌 11 class PlantCard{ 12 //植物编号 13 int index; 14 //植物名字 15 string name; 16 //价格 17 int price; 18 //冷却时间 19 int CD; 20 //是否选中 21 bool flag; </pre>	<pre> 27 public: 28 //设定参数 29 void set(int i, const string& iname, int iprice, int iCD){ ... } 30 //输出植物购买信息 (名字、价格, 以及冷却进度) 31 void print(); 32 //切换选中/非选中 33 void setSelect() { flag = true; print(); } 34 void setUnSelect() { flag = false; print(); } 35 //冷却 36 void cooling(); 37 //是否冷却结束 38 bool coolingDone(); </pre>
--	---

d) Store 类

Store 类表示“商店”，处理游戏中和植物购买的相关的操作。比如展示植物卡片、记录已有阳光、自然掉落阳光等等。这里的核心是所有植物的卡片构成的一个一维数组。

部分数据成员及成员函数展示：

```
48 class Store
49 {
50     //阳光数量
51     int sun;
52     //自然生产阳光的速度
53     int speed;
54     //可种植的植物
55     PlantCard plants[PLANT_TYPE_MAX];
56
75     //商店初始化
76     void init();
77     //刷新阳光数量的输出
78     void refreshSun();
79     //增加阳光
80     void addSun(int isun=50) { sun += isun; refreshSun(); }
81     //购买，扣除阳光，返回是否购买成功
82     bool pay(int choice, int x, int y, Map &map);
83     //商店运行(周期增加阳光、冷却植物等)
84     void run();
85     //金手指：瞬间完成所有冷却(便于测试)
86     void renew();
```

由于 PlantCard 类提供了一个 set 函数用于设定植物卡片的信息，所以这里可以很方便的统一设定所有植物的信息：

```
59 public:
60     Store() {
61         plants[0].set(0, "向日葵", 50, 75);
62         plants[1].set(1, "豌豆射手", 100, 75);
63         plants[2].set(2, "窝瓜", 50, 200);
64         plants[3].set(3, "樱桃炸弹", 150, 200);
65         plants[4].set(4, "坚果墙", 50, 200);
66         plants[5].set(5, "寒冰射手", 175, 75);
67         plants[6].set(6, "双发射手", 200, 75);
68         plants[7].set(7, "大蒜", 50, 75);
69         plants[8].set(8, "火爆辣椒", 125, 300);
70         plants[9].set(9, "土豆地雷", 25, 200);
71         plants[10].set(10, "地刺", 100, 75);
72         plants[11].set(11, "高坚果墙", 125, 200);
73     }
```

编号 价格 冷却时间(s)*10
(例如第1个表示 7.5s)

〈关于“时间”表示〉在本项目中，与时间相关的值往往都是使用×10之后的值(比如上面的7.5s的冷却时间，输入的是75)，为何要这样？

首先，这里的时间的值，最终都是会被计算转化成多少个时钟周期来使用，这里不适合使用浮点数类型表示时间(没必要使用，且容易引起不精确等等的其他问题)，但是如果时间都是整数秒，精度往往不够，因为有时候又会需要带小数的时间，比如0.5秒这样的。而假设时钟周期是100ms，那其实是可以做到时间粒度为0.1s。

有些抽象，这里用一个实例来说明，且看下面这句代码(Map.h文件中，位于PlantCard类的set函数中)。常量SLEEP_TIME的值为时钟周期的毫秒数(项目中值定为100ms)，变量CD表示(iCD/10)秒这个时间量对应多少个时钟周期。本来1000/SLEEP_TIME就表示1秒对应多少个时钟周期，x秒就是x*1000/SLEEP_TIME。而这里是iCD的值是时间秒数×10，所以也要多除一个10。这样一来，利用c++整数乘除法的规则，这个计算公式利用避免了小数的运算，但是也算出了小数时间对应的时间周期数。

```
33 CD = iCD * 1000 / 10 / SLEEP_TIME; //(iCD/10)的值表示多少秒的CD时间
```

e) Game 类

这里整个游戏的核心就是 Game 类，它的两个大部分就是上面已经提到了的 Map 类对象和 Store 类对象。

```

12 class Game {
13     Map map;
14     Store store;

```

这里还有很重要的部分就是：

```

16     //子弹队列 (豌豆射手等发出的)
17     list<Bullet*> bullets;
18     //僵尸队列
19     list<Zombie*> zombies;

```

这里子弹和僵尸都是需要执行移动操作的，且会消亡，需要删除，所以在 Game 类中使用 list 容器来组织。

注：为什么植物不在 Game 类中组织？对于植物来说，植物种植在“格子”中，且每个格子只有一个植物，所以植物这一结构存放在的是 Grid 类中。（当然，“格子”也需要访问到本格子内的僵尸，所以其实“格子”保留了僵尸类对象的指针的副本，但对于僵尸而言，真正的组织结构其实还是在 Game 类的 list 中）。

Game 类中，记录了当前游戏的不同状态，以及有不同状态下使用的按键识别函数。

```

21     //当前动作状态, 可选择的值包括在下面的枚举类型中
22     int state;
23     //状态: 正常、购买状态、铲除植物、游戏暂停
24     enum GAME_STATE { NORMAL, STORE, SHOVEL, PAUSE };
25     //选择铲除植物中
26     void weeding();
27     //选择购买植物中
28     void shopping();

```

也有着许多其它的辅助实现游戏逻辑的函数，如下图。

```

39     //铲除某个位置的植物
40     void delPlant(int ix, int iy);
41     //子弹移动
42     void moveBullet();
43     //子弹的显示输出
44     void printBullet();
45     //指定位置增加某个类型的僵尸
46     void addZombie(int x, int y, int type);
47     //僵尸产生速度
48     int make_speed;
49     int make_counter;
50     //僵尸产生逻辑
51     void makeZombies();
52     //僵尸移动
53     bool moveZombie();
54     //清除无效僵尸 (血量<=0)
55     void clearZombie();

```

f) Bullet 类

Bullet 类表示游戏中射手型植物发射的子弹。子弹逻辑也稍复杂，所以单独拿出来写了一个类，而且也便于实现两种子弹：普通豌豆、冰冻豌豆。

下图是最基本的子弹，普通豌豆。（成员变量为 protected 类型，便于派生类使用）


```

7  class Bullet
8  {
9      protected:
10         //子弹当前的位置
11         int x, y;
12         //移动速度 (speed个时钟周期移动一个字符格子)
13         int speed;
14         //攻击力 (每次击中僵尸, 扣除的血量)
15         int attack;

```

```

20     public:
21         Bullet();
22         //计算并设置子弹的起始坐标 ((dx,dy)位置的植物发射子弹)
23         void setXY(int dx, int dy);
24         //向前移动
25         void move(Map &map);
26         //绘制子弹
27         virtual void paint();
28         //攻击僵尸
29         virtual void hitZombie(vector<Zombie*> &zombie);
30         //是否击中
31         bool hit;
32     };

```

而对于冰冻豌豆(SnowBullet 类), 这里继承了 Bullet 类, 并重写了两个方法 (基类中是虚函数), 因为冰冻豌豆本身的绘制、攻击效果与普通豌豆不同 (击中僵尸后, 会使得僵尸开启冷冻状态, 攻击/移动速度减半, 有报纸/铁门的僵尸除外), 而其他则一模一样。

```

34     //寒冰射手的子弹
35     class SnowBullet :public Bullet {
36     public:
37         void hitZombie(vector<Zombie*> &zombie);
38         void paint();

```

g) Plant 类

这是所有植物的基类。包括了一些植物的基本属性和方法。(之后介绍 12 种具体植物的设计细节)

属性:

```

11  /* 所有植物的基类 */
12  class Plant {
13      protected:
14         //植物名
15         string name;
16         //所在网格
17         int x, y;
18         //植物耐久度 (默认都是300)
19         int HP;
20         int maxHP;

```

```

29         //植物是否可以被吃, 如: 地刺不被吃
30         bool eatable;
31         //植物是否可被跳过 (高坚果强不行)
32         bool skipable;

```

方法:

```

34 //返回植物名字
35 const string& getName() const { return name; }
36 //设定所在网格
37 void setXY(int ix, int iy) { x = ix; y = iy; }
38 //特殊行为（默认无），如：发射豌豆、产生阳光
39 virtual void go(Game& nowGame) {}
40 //输出植物名
41 void printName();
42 //植物名的颜色
43 int nameColor;
44 //输出剩余生命值（默认格式输出，可重写，比如地刺剩余HP）
45 virtual void printLife();
46 //额外信息输出（默认无），如：土豆雷的装填信息
47 virtual void printExtra() {}
48 };

```

其中，虚函数 go 是具体各种植物发挥作用的关键函数（派生类通过重写这个方法来实现各个植物的特性）。在 Game 类的 loop 函数（游戏主循环）中，每个时钟周期它都会被调用。

h) Zombie 类

Zombie 类表示最普通的僵尸，拥有着默认的属性，也提供设置属性的方法，以及其他的一些所有僵尸共通的方法。（之后介绍 11 种具体僵尸的设计细节）

部分属性：

<pre> 11 //普通僵尸 12 class Zombie 13 { 14 protected: 15 //所在网格 16 int x, y; 17 //僵尸名 18 string name; 19 //移动速度 20 int speed; 21 //血量 22 int HP; //当前 23 int maxHP; 24 //攻击力 25 int attack; 26 //击杀得分 27 int score; </pre>	<pre> 29 //属性参数设定 30 void set(const string& str, int iscore, int 31 32 //用于移动的计数 33 int counter; 34 //标记处于吃植物的状态 35 bool eating; 36 37 //冰减速效果：移动/攻击速度减半 38 bool freezing; 39 //减速效果持续时间 40 int freezing_time; 41 //时间计数 42 int freezing_counter; 43 44 //遇到大蒜 45 bool scape_flag; 46 //多少秒后移动到其他行 47 int scape_time; 48 //时间计数 49 int scape_counter; 50 </pre>
---	---

一些方法：

```

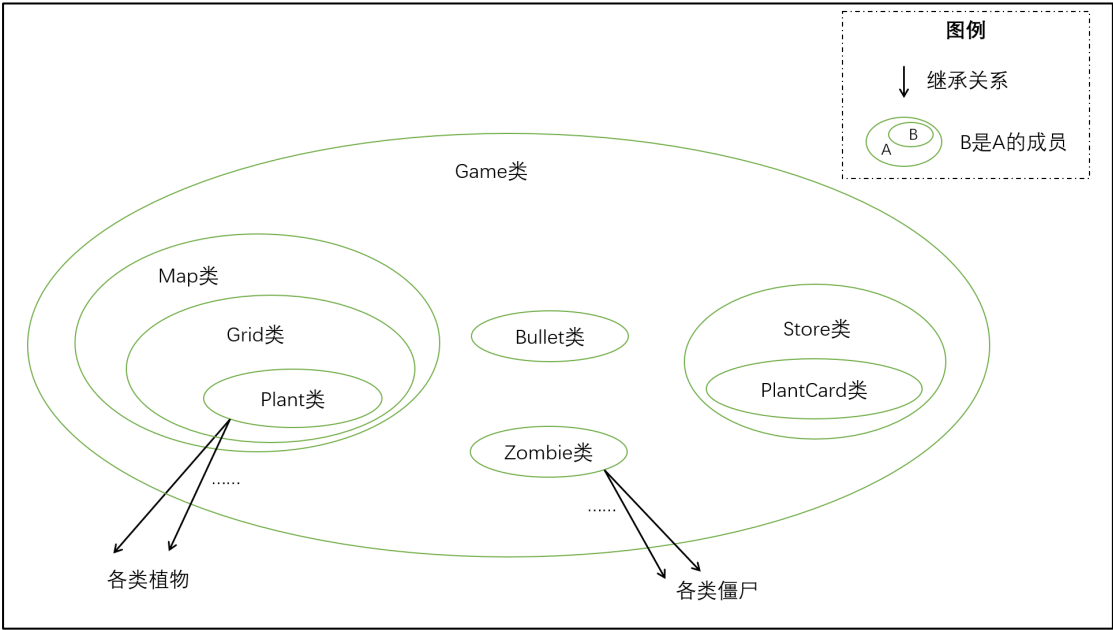
51 //输出僵尸名字（如：冰冻状态名字会变色）
52 virtual void printName();
53 //额外信息输出（有无铁门/报纸等信息）
54 virtual void printExtra() {}

```

```
76 //设定起始位置
77 void setXY(int ix, int iy) { x = ix; y = iy; }
78 //僵尸行动 (包括位移、吃植物等动作), 返回值代表是否攻破
79 virtual bool move(Map& map);
80 //僵尸特殊效果 (默认无)
81 virtual void go(Game &nowGame) {}
82 //被攻击
83 virtual void hit(int attack) { HP -= attack; }
84 //开启被减速的效果 (铁门僵尸等可防御)
85 virtual void setFreezing() { freezing = true; freezing_counter = 0; };
86 //开启大蒜效果 (准备随机切到相邻的行)
87 void setScape() { ... }
```

这里的 go 函数类似于植物一派中的 go 函数，也是会在 Game 中的 loop 函数中每个时钟周期被调用。只不过大部分僵尸不需要用到，所以函数体为空就行，少数有特殊效果的僵尸，比如舞王僵尸和小丑僵尸需要用到，则重写即可。

2) 综上所述，各个类之间的关系大致如下图所示：



3) 12 种植物的设计细节

12 种植物属性总结如下表：

植物名	购买花费	购买冷却时间 (秒)	HP	攻击力	攻击/生产间隔/准备时间 (秒)
向日葵	50	7.5	300	0	8
豌豆射手	100	7.5	300	20	1.4

窝瓜	50	20	300	1800	0.8
樱桃炸弹	150	20	300	1800	0.5
坚果墙	50	20	4000	0	-
寒冰射手	175	7.5	300	20	1.4
双发射手	200	7.5	300	20（单发）	1.4
大蒜	50	7.5	800	0	-
火爆辣椒	125	30	300	1800	0.6
土豆地雷	25	20	300	1800	7.5
地刺	100	7.5	300	20	1.0
高坚果墙	125	20	8000	0	-

总体来说，各种具体植物的实现有一定的相通之处，比如都是需要在派生类的构造函数种调用基类的 set 函数，设定植物属性。然后，根据具体的特性去重写对应方法。

```

22 //设定植物属性: 名字、HP (默认300)
23 void set(const string& str, int ilife = 300) {
24     name = str; maxHP = HP = ilife; eatable = true; skipable = true; nameColor = DEFAULT_COLOR;
25 }

```

下面分别介绍各类植物的设计细节：

①向日葵

额外的属性如下图（8 秒生成一次阳光）。且重写了 go 函数，在其中实现了周期性的增加阳光的功能。

```

67 //向日葵
68 class SunFlower :public Plant {
69     //产生阳光的速度
70     int speed;
71     //时钟计数
72     int counter;
73 public:
74     SunFlower() {
75         set("向日葵"); //HP默认300
76         speed = 80 * 1000 / 10 / (SLEEP_TIME); counter = 0; //8秒
77         nameColor = PLANT_STATIC_COLOR;
78     }
79     void go(Game& nowGame);
80 };

```

<对“nameColor”的说明>

这里为了使得控制台下显示效果更好一些，也便于植物与僵尸可以鲜明地区分开，这里对植物的名字输出采用了“色块”的方式，由于控制台的颜色支持得不多，且好看的颜色更少，所以这里将植物分三类颜色显示，分别用以下常量表示：

PLANT_ATTACK_COLOR、PLANT_STATIC_COLOR、PLANT_BOMB_COLOR

分别表示攻击型植物（豌豆射手、寒冰射手、双发射手）、静态型植物（向日葵、大蒜、坚果墙、高坚果墙）、炸弹型植物（窝瓜、土豆地雷、樱桃炸弹、火爆辣椒）。地刺则特殊，采用默认的黑底白字，无色块。

显示效果分别如下：



②豌豆射手、寒冰射手、双发射手

首先，是最典型的豌豆射手。这里重写了 go 函数，实现了周期性发射豌豆的功能。

```
53 //豌豆射手
54 class PeaShooter :public Plant {
55     //发射速度
56     int speed;
57     //时钟计数
58     int counter;
59 public:
60     PeaShooter() {
61         set("豌豆射手"); //HP默认300
62         speed = 14 * 1000 / 10 / (SLEEP_TIME); counter = 0; //1.4秒
63         nameColor = PLANT_ATTACK_COLOR;
64     }
65     void go(Game& nowGame);
66 };
```

注：这里为什么没有攻击力这个属性？攻击力其实体现在“子弹”上，所以这里豌豆射手不存在攻击力这一属性，而是“子弹”Bullet 类存在攻击力属性，豌豆射手只负责产生“子弹”。

同理，寒冰射手也是基本类似，只是周期性生产的“子弹”不是普通的 Bullet 类对象，而是 SnowBullet 这个派生类对象。如下图：

```
159 Bullet *p = new SnowBullet;
```

而双发射手几乎与豌豆射手一样，只是在重写的 go 函数中，每个周期要产生两个子弹，同时要注意两个子弹的间距要恰当，否则显示效果不好。

③坚果墙、高坚果墙

坚果墙实现较为简单，因为没有特殊效果，只需要设置 HP 为 4000，不需要重写 go 函数。

高坚果墙类似，HP 设置为 8000，但同时要将 skipable 成员变量置为 false，这样就使得撑杆跳僵尸无法越过。

④地刺

地刺与其他植物有较大的不同。

首先是名字显示设置的不同，不带色块；然后，要将 eatable 成员变量置为 false，这样所有的僵尸都不会停下来吃它；其次，它有攻击性，重写 go 函数，在其中实现周期性的对本格子内的所有僵尸产生伤害，并且产生伤害的时候本格子红色闪烁一下；最后，还重写了 printLife()函数，因为地刺不需要输出 HP 值。

⑤窝瓜、土豆地雷、樱桃炸弹、火爆辣椒

这四类植物一定程度上都是“炸弹类型”，攻击力都是 1800（目前没有僵尸可以承受一击）。

首先，窝瓜的设计是对附近一个格子内的所有僵尸造成 1800 点伤害，且自身消亡。这里为了使得效果接近原版，这里窝瓜要 0.8s 后才产生效果（这段时间中，窝瓜所在格子会有红色闪烁）。0.8s 后，如果附近不存在僵尸，等待直到僵尸靠近就产生效果。

```
83 //窝瓜
84 class Squash :public Plant {
85     //攻击力
86     int attack;
87     //种植时间延后开始产生效果
88     int speed;
89     //时间计数
90     int counter;
```

同理，土豆地雷则设定一个较长的装填时间（还重写了 printExtra()方法，所以会额外显示装填进度），大约 7.5s，装填完毕后（所在格子会开启红色闪烁）与窝瓜效果一致。

而对樱桃炸弹来说，这里设计是种植一段时间后（0.5s，这段时间中爆炸范围内的格子红色闪烁），不管有无僵尸，立即对周围九个格子内造成 1800 点伤害，且自身消亡。火爆辣椒同理，只是爆炸范围变成所在的一行。

⑥大蒜

大蒜也是重写了 go 函数，在其中，会调用同一个格子内的僵尸对象的 setScape 方法，使其开启“大蒜效果”。而“大蒜效果”的具体实现由 Zombie 类完成，其行为是一段一段时间后，随机切换到另一行。

4) 11 种僵尸的设计细节

11 种僵尸的属性总结如下：

僵尸	击杀得分	HP	速度（秒/格）	攻击力（点伤害/秒）
普通僵尸	50	200	4.5	100
摇旗僵尸	50	200	3	110
路障僵尸	75	570	4.5	120
铁桶僵尸	125	1300	4.5	120
橄榄球僵尸	175	1600	2.5	120
铁门僵尸	125	1300	4.5	120
读报僵尸	125	300	4.5/1.2	100/200
撑杆僵尸	100	430	2/4.5	100
小丑僵尸	250	800	1.2	100
舞王僵尸	350	500	1.2/5.5	150
伴舞僵尸	50	200	5.5	100

（斜杆后的值表示第二个状态下的属性值，比如：撑杆僵尸跳前是 2 秒/格，跳完后是 4.5 秒/格）

下面分别介绍各类僵尸的实现细节：

①普通僵尸

与 Plant 类不同，僵尸的基类 Zombie 类是有实际含义的，它表示普通僵尸，属性都是默认参数。它提供了 set 函数用于设定属性参数，如下：

```
29 //属性参数设定
30 //名字、击杀得分、HP
31 //速度: (ispeed/10) 秒/格
32 //攻击力: (iattack) 点伤害/秒
33 void set(const string& str, int iscore=50, int ilife = 200, int ispeed = 45, int iattack = 100) {
```

②摇旗僵尸、路障僵尸、铁桶僵尸、橄榄球僵尸

这四类僵尸，在行为上与普通僵尸一致，只是属性参数设置得不同。所以只需要在派生类的构造函数中调用下 set 函数设置相应的参数即可。例如：

```

116 class Football_Zombie :public Zombie {
117 public:
118     Football_Zombie() { set("橄榄球僵尸", 175, 1600, 25, 120); }
119 };
120

```

击杀得分
 速度 2.5秒/格
 HP
 攻击力 120点伤害/秒

摇旗僵尸仅是普通僵尸稍快，且总是每波僵尸第一个刷出（在 Game 类中的 makeZombies 函数中实现）；路障僵尸、铁桶僵尸都是 HP 高，速度与普通僵尸一样；橄榄球僵尸是速度快且 HP 高。以上三种僵尸均比普通僵尸伤害略高。

③铁门僵尸

属性与铁桶僵尸一致，但铁门僵尸在有铁门状态下，被寒冰射手击中不会进入冰冻效果（攻击/移动速度减半）。这里按 HP 值划分，铁门僵尸 HP 低于 500 时，表示铁门被打破。

为了实现这个特性，这里重写了 setFreezing 函数（被寒冰豌豆击中后被调用），在新的实现中会先判断有无铁门，再是否进入冰冻状态。

同时，这里也重写了 printExtra 函数（需要输出额外信息时被调用，基类中函数体为空），因为铁门僵尸有“有铁门”和“无铁门”两种状态，需要作为额外的信息输出。

④读报僵尸

读报僵尸 HP 小于 200 时，表示报纸被打烂，进入红眼状态，此时速度变得很快，攻击翻倍。同时，他像铁门僵尸一样，有报纸的状态下，免疫寒冰。

所以，就像铁门僵尸，这里也是重写了 setFreezing 函数以及 printExtra 函数。

但额外地，还重写了 hit 函数（僵尸受伤扣血时被调用），因为这里要根据 HP 的变化，改变属性参数，所以要监测 HP 的变化；也重写了 printName 函数，因为红眼状态下，名字变红。

⑤撑杆僵尸

撑杆僵尸有两种状态：有/无撑杆，所以重写了 printExtra 函数。

而撑杆僵尸的行进逻辑与普通僵尸不同，所以这里重写了 move 函数（僵尸的行动逻辑），调整其逻辑为越过第一个可以吃的植物（所以地刺不阻挡它），但不能越过 skipable 属性为 false 的植物（如高坚果墙）。且越过后，速度减慢。

⑥小丑僵尸

小丑僵尸与其他僵尸均不同，他有自己的特殊效果：爆炸（主要是因为这个效果会影响到其他对象，其他“格子”，而不是仅仅是影响自己，所以特殊）。所以这个时候，就需要重写 go 函数（每个时钟周期都会被调用，且传入的参数为当前 Game 类对象的引用）。

所以，小丑僵尸的 go 函数逻辑为：有个计时器，每 2 秒种决策一下，以 2/3 的概率决定要不要自爆。爆炸波及周围 9 格子的植物，格子红色闪烁一下，自己也会消亡。

⑦舞王僵尸、伴舞僵尸

如同小丑僵尸，舞王僵尸有自己的特殊效果，所以也是重写了 go 函数，其中的逻辑为：出现 4 秒后，在其上下左右四个格子各召唤出一个伴舞僵尸，召唤完成后，速度变慢。且这里为了平衡游戏难度，只给了舞王僵尸一次的召唤机会，即用了个变量来记录是否已召唤过。

而伴舞僵尸，属性与普通僵尸基本一致，只是速度略慢（但和召唤后的舞王僵尸速度是一致的），且不被随机刷出，只可能被舞王僵尸召唤出。

三、程序的功能特点和运行操作方法

<关于运行的一些说明>

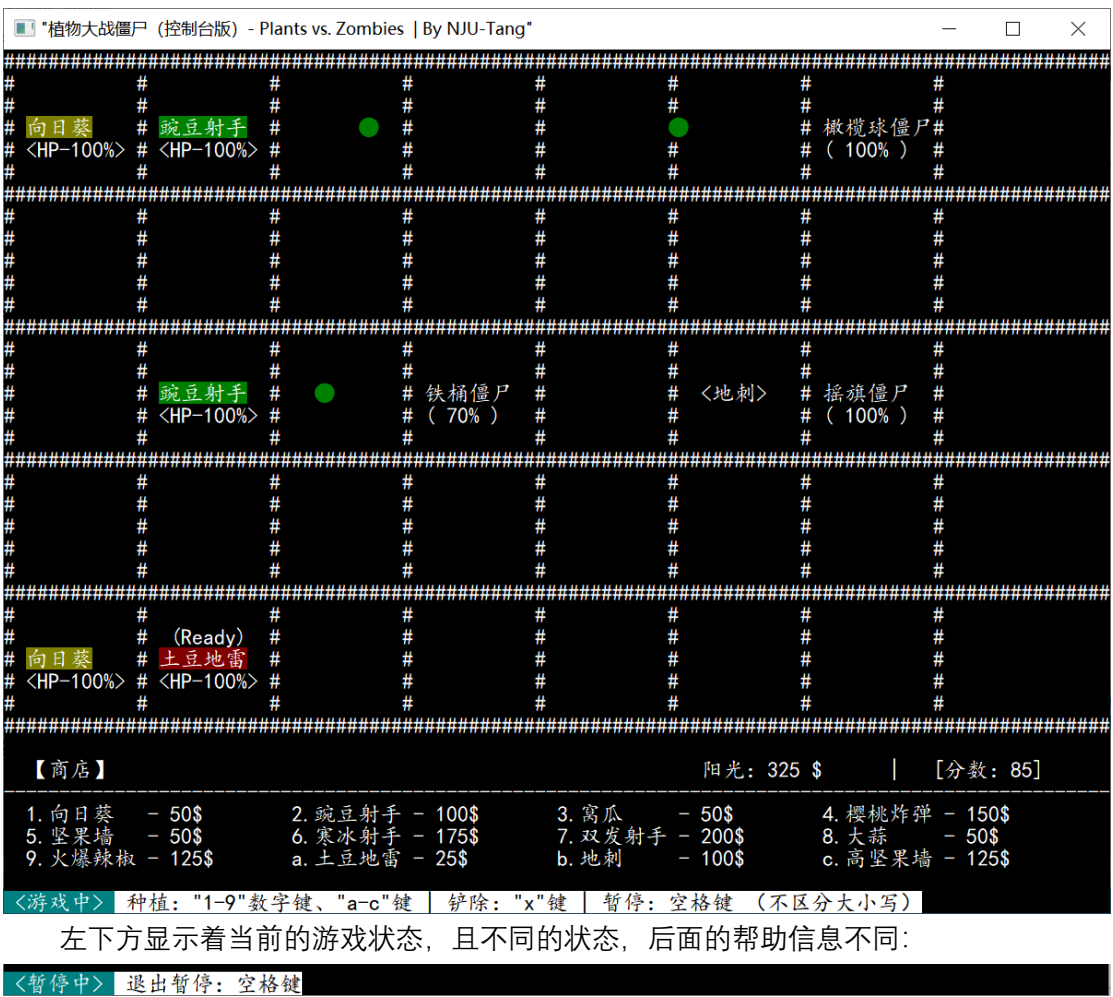
- ① 若直接双击运行“可执行文件”目录下的PVZ.exe文件，在默认使用搜狗拼音输入法（中文）的电脑上，发现底部会多出“搜狗拼音输入法...”信息，而整体坐标也会出现错位。但经过测试发现，在这种情况下，使用cmd命令行运行PVZ.exe则不会有这个问题。所以，若出现这种情况，解决方法可参考：在“可执行文件”目录下，按住shift键，按下鼠标右键，选择“在此处打开命令窗口”，然后输入PVZ，回车运行。
- ② 若使用“工程项目目录”下的项目源文件编译生成，请使用vs2017,经自己测试，发现vs2013编译生成会报错。

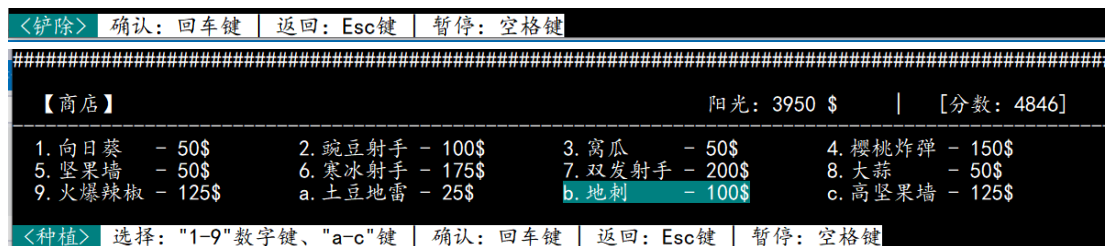
首先，程序目录下只有一个.exe可执行文件，无其他依赖文件，运行即可开始游戏。

说明：推荐打开.exe文件后，窗口上方标题栏处右键，进入“属性”-“字体”设置页面，设置字体为“楷体”，不勾选粗体，大小为14(或者16)，以便获得较好的显示效果（如以下截图）。

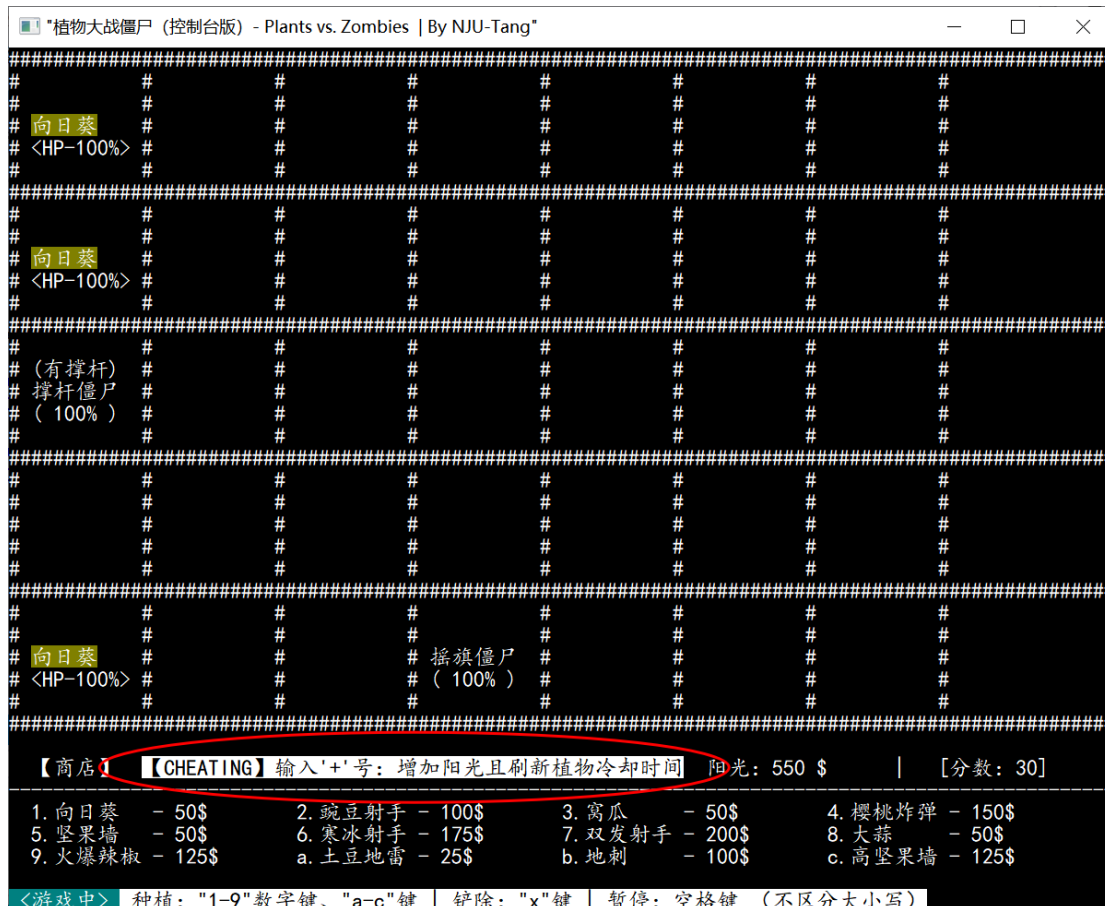
开始游戏后，每16秒产生一波僵尸，每波僵尸必有一个摇旗僵尸。一开始是每波僵尸有一只随机种类的僵尸，后续，随着玩家分数的增加，每波僵尸的数量逐渐增加，难度递进。具体表现为，每增加2000分，每波僵尸数量加一。

游戏界面如下：

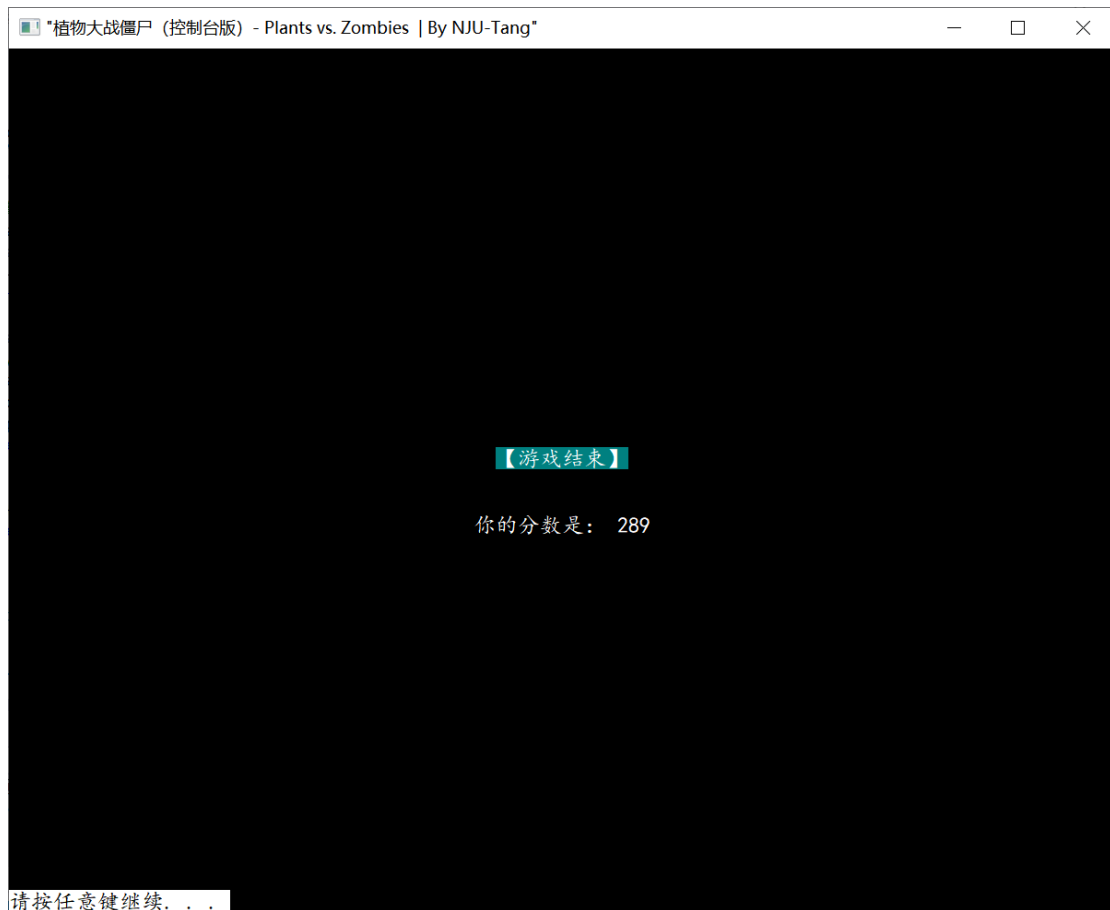




在僵尸快攻破的时候，输出 CHEATING 提示信息：（游戏之初不建议使用，破坏游戏体验，所以只会在僵尸快攻破时，才出现此提示；设计之初的目的是便于调试植物...后来为了平衡游戏难度保留了下来）



游戏结束画面，输出最终分数：



四、代码实现中值得一提的地方

1) 画面刷新机制

这里的 Grid 类负责的显示内容多种多样, 涉及到的元素很丰富, 输出逻辑也比较复杂, 一旦有内容变化, 就需要刷新显示新的内容 (比如僵尸扣血、植物铲除等等的操作)。然而如果过于频繁地清空格子内容、输出新内容, 整个画面就会闪烁得很厉害。

所以, 这里使用了一点技巧来管理 Grid 类的输出。

在 Grid 类的内部, 这里使用了一个 bool 变量来记录本次时钟周期内是否需要刷新显示, 如果需要, 则刷新, 否则, 不刷新。且每次刷新将这个变量归位。

所以, 这样一来, 就相当于把一个时钟周期内的刷新操作, 集中在了时钟周期的尾部, 也就是一个周期内最多只刷新一次。这样减少了刷新的次数, 同时也有利于不同层次的元素的输出 (也就是可以先统一输出完 Grid 的内容, 再统一输出一遍“子弹”, 因为“子弹”要覆盖 Grid 的内容)。

2) 子弹飞行

由于这里的子弹“●”占两个字符的宽度，所以子弹一个一个字符格子的移动时，就会出现错位问题，比如：子弹的右半边与一个中文字符的左半边重叠的时候，此时右侧的一整行字符会出现错位（整体向左一个字符格）。


（后续与同学讨论，以及大量试验，发现“●”与中文字符颜色相同的情况下才会出现这个错位的情况，而奇怪的是，只要颜色不同就不会，可实现“●”的右半边遮住中文字符的左半边，原因无解...）

这里，采用的解决方法是：子弹“●”每次移动两个，且通过设计格子加边界的宽度为偶数，这样一来，子弹“●”每次要么完整地遮住一个中文字符，要么不遮住，而不再会出现一半与一半重叠的现象。

3) 复用基类代码

这里大量使用了继承的思想，而有的时候，复用基类的代码可节省大量的精力。比如，这里的撑杆僵尸，因为其移动的逻辑的不同，所以这重写了 move 函数，而撑杆僵尸跳完之后，其实就是按普通僵尸的逻辑移动，所以在重写的 move 函数中，在跳完分支情况下，可以直接调用基类的 move 函数，如下图：

```
145     else { //否则，调用普通僵尸（父类）的行进逻辑即可
146         return Zombie::move(map);
147     }
148 }
```



全文结束，感谢阅读！*Thanks For your time...*

—— 课程设计二 姓名：唐金麟 TangJinlin@smail.nju.edu.cn