第二次课程设计报告

植物大战僵尸 (控制台版) —— 完整版

计算机科学与技术系

191220008 陈南疃

一、概要

1、实验内容

模仿塔防游戏《植物大战僵尸》，实现基于Windows控制台运行的简易版植物大战僵尸，对其部分核心功能进行复现，需要实现的场景：

- 前院场景（纯草地无水池）
- 白天（系统会产生自然光）
- 无尽模式（需要记分牌）

2、游戏逻辑：

- 按照一定的策略随机产生僵尸，从马路进入玩家的庭院，吃掉玩家种植的植物，以庭院左边的底线为目标前进。
- 玩家通过收集阳光、种植植物反击以攻击消灭僵尸并保护房子。
- 游戏失败：任何一只僵尸进入了庭院左边的底线。
- 游戏胜利：由于是无尽模式所有没有胜利条件，目标是能够持续抵挡僵尸的进攻，已获得更多的累计积分。

3、已经实现的内容

（1）整体的UI设计

- 花园：5行9列
- 商店：商品、阳光数、分数
- 提示信息：当前可进行的操作

（2）植物

- 豌豆射手
- 向日葵
- 双发射手
- 寒冰射手
- 坚果墙
- 高坚果

- 窝瓜
- 樱桃炸弹
- 大蒜
- 南瓜头

（3）僵尸

- 普通僵尸
- 路障僵尸
- 读报僵尸
- 撑杆僵尸
- 小丑僵尸
- 投石僵尸

（4）完善的游戏逻辑

- 见类的设计部分

二、主要的类的设计

1、模块划分

根据植物大战僵尸的游戏设计，可以大致归纳出有以下几个部分：

- UI界面
- 游戏

  ① 商店

  ② 植物商品

  ③ 花园

  ④ 植物

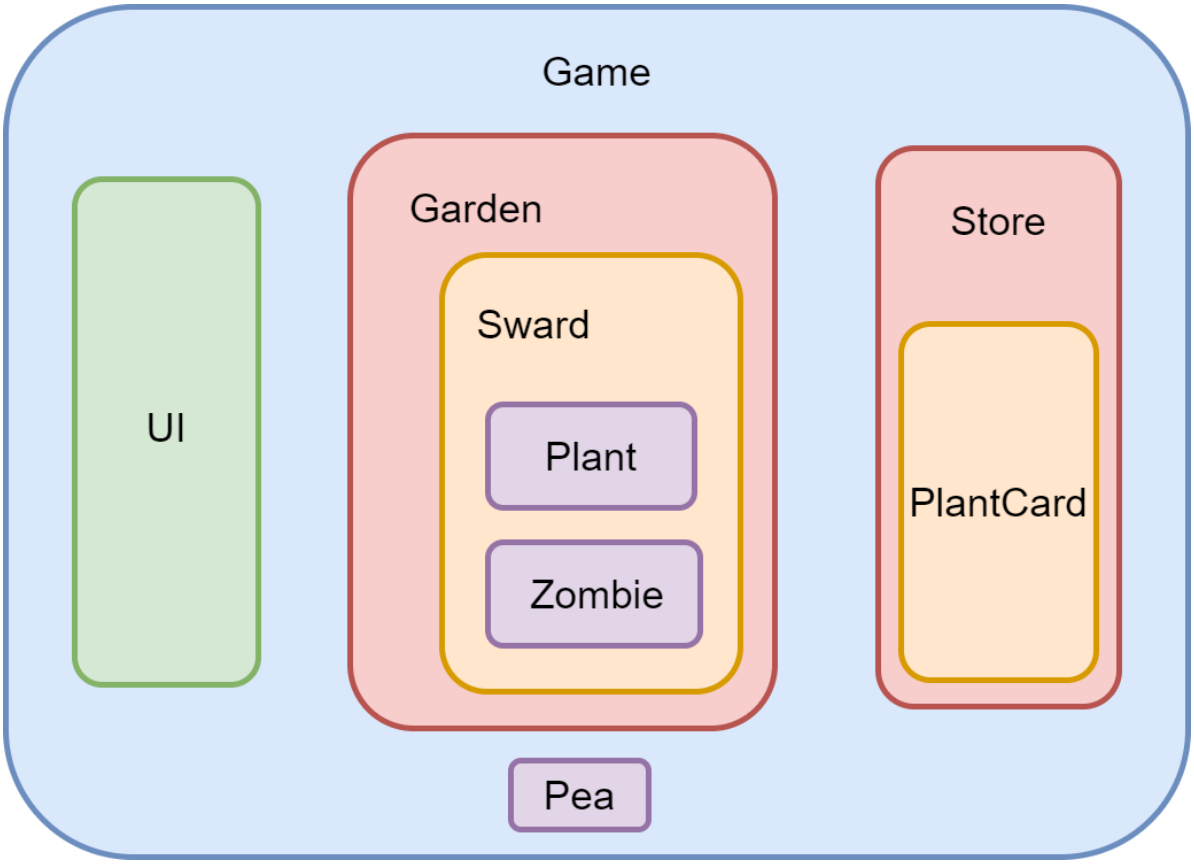
  ⑤ 僵尸

  ⑥ 豌豆

据此，创建了以下头文件及其定义的类：

- UI.h : class UI
- Game.h : class Game
- Store.h : class Store, class PlantCard
- Map.h : class Garden, class Sward
- Plant.h : class Plant 和它的派生类
- Zombie.h : class Zombie 和它的派生类
- Pea.h : class Pea 和它的派生类

类与类之间的关系如下：



2、类的介绍

（1）UI类

```
class UI
{
public:
    UI();
    static void Set_Window(); // 设置窗口相关参数
    static void Hide_Cursor(); // 隐藏光标
    static void Move_Cursor(const int x, const int y); // 设置光标位置
    static void Set_Color(int color); // 设置颜色
    static void Print_With_Color(const string& str, int color); // 带颜色的字符串输出
    static void Print_With_Color(int num, int color); // 带颜色的数字输出
};
```

UI类负责对游戏界面进行设置，使得控制台窗口大小合适，窗口标题自定义，游戏过程中没有光标显示，可以在控制台任意位置进行带有颜色的打印操作。

类中的成员函数都被设置成为了静态static，一是因为这些函数没有用到其他的类，二是为了方便在其他类中随时调用，以便进行打印操作。

(2) Game类

```
class Game
{
private:
    UI ui; // 当前UI
    Garden garden; // 当前花园
    Store store; // 当前商店
    STATE state; // 游戏状态
    int num_x; // 选择的草地的x坐标
    int num_y; // 选择的草地的y坐标
    vector<Pea*> peas; // 所有豌豆
    int plant_type; // 选择的植物种类
    int score; // 分数
    bool score_update_flag; // 是否需要更新分数
    bool tip_update_flag; // 是否需要更新提示信息
    int timer; // 计时器
    int zombie_speed; // 僵尸产生速度
public:
    Game();
    void Menu(); // 主菜单
    void Game_Start(); // 开始游戏
    void Game_Over(); // 游戏结束
    void Quit(); // 退出游戏

    void Running(); // 正常运行
    void Perchasing(); // 购买植物
    void Weeding(); // 移除植物
    void Pausing(); // 暂停游戏

    void Add_Pea(Pea* pea); // 添加豌豆
    void Delete_Pea(Pea* pea); //删除豌豆

    void Tip(); // 显示提示信息

    void Create_Zombie(); // 生成僵尸

    void Update_Score(); // 更新分数

    ...
};
```

Game类是游戏的核心逻辑部分，在Game类中定义游戏中的三大分支类的对象：UI类，Store类和Garden类。

游戏整体是一个状态机，有如下四个状态：

```
enum STATE { RUNNING, PERCHASING, WEEDING, PAUSING }; // 游戏运行状态
```

而Game类中则提供了每个状态对应的函数：

```
void Running(); // 正常运行
void Perchasing(); // 购买植物
void Weeding(); // 移除植物
void Pausing(); // 暂停游戏
```

在Game类中还存储了游戏中所有的豌豆

```
vector<Pea*> peas; // 所有豌豆
```

由于豌豆在植物射出后便不再受植物管理，而是成为一个单独的逻辑对象，所以放在Game类中统一管理，而不是由Plant类或Sward类管理。

但植物和僵尸并没有放在Game类中统一管理，是因为每个植物或僵尸都是属于某一块草地的，由Sward类管理更为合适，而豌豆需要频繁的生成、消亡，并在不同的草地中不断切换，如果由Sward管理则会显得有些繁琐，所以由Game类管理会更合适。

(3) Store类和PlantCard类

```
class Store
{
private:
    int sunshine; // 阳光数
    int sunshine_speed; // 自然阳光产生速度
    int sunshine_produce; // 单次产生的自然阳光数
    bool sunshine_update_flag; // 是否需要更新阳光数
    int timer; // 计时器
    PlantCard plant_card[PLANT_TYPE_MAX]; // 商品序列
public:
    Store();
    void Print(); // 打印商店
    bool Perchase(int plant_type, int num_x, int num_y, Garden& garden); // 购买商品
    Plant* Type_To_Plant(int plant_type, int num_x, int num_y); // 商品序号对应植物
    void Sunshine_Produce(Game& game); // 产生自然阳光
    void Update_Sunshine(Game& game); // 更新阳光数

    ...
};
```

Store类表示商店的类，存储了所有的商品，并能对其进行购买；还存储了拥有的阳光数，并能自动产生自然阳光。

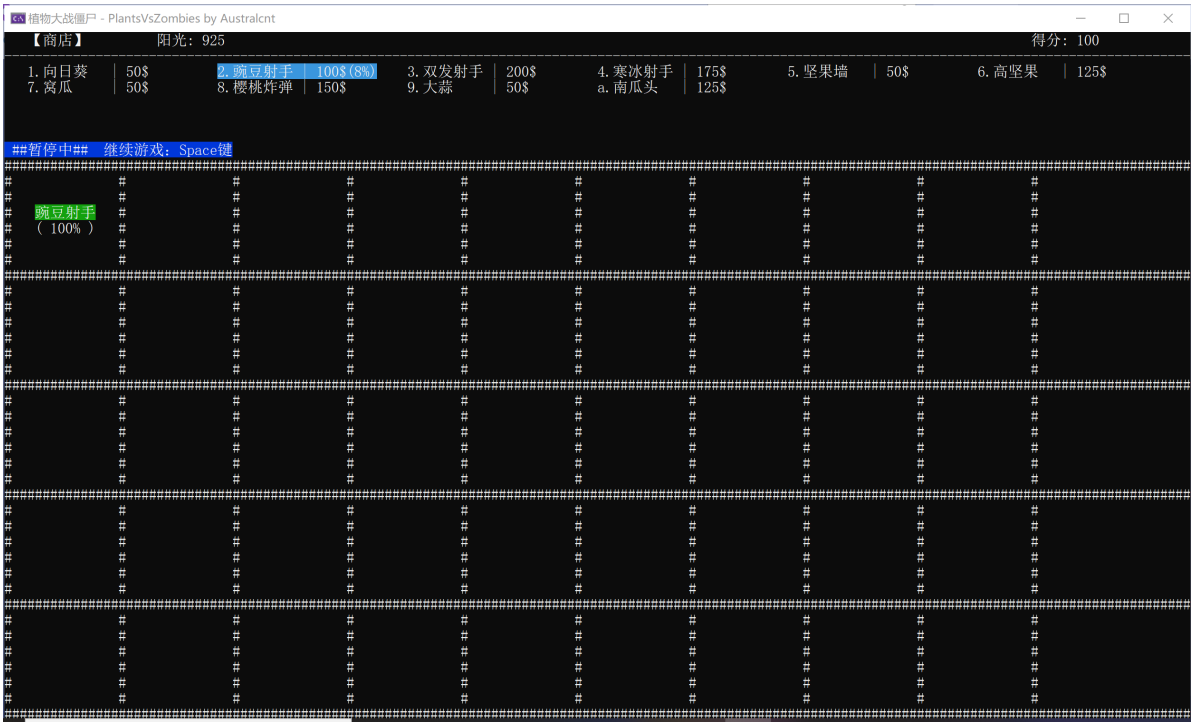
由于商品种类是固定不变的，所以不用vector存储，而是采用一维数组进行存储。

```
class PlantCard
{
private:
    int index; // 商品序号
    string name; // 商品名字
    int price; // 商品价格
    int cd; // 商品冷却时间
    int timer; // 计时器
    bool select_flag; // 是否被选中
    bool cooldown_flag; // 是否正在冷却
    bool update_flag; // 是否需要更新商品信息
public:
    void Set(int _index, string _name, int price, int cd); // 设置商品相关参数
    void Print(); // 打印商品
    void Cool_Down(); // 冷却商品
```

```
    ...
};
```

PlantCard类则是对应了每一个商品。每个商品都有其对应的序号、名字、价格、冷却时间。商品在创建Store类对象的同时，设置好每一种商品。

当购买某商品后，该商品会进入冷却，商店中会展示商品冷却时间百分比和冷却颜色：



(4) Garden类和Sward类

```
class Garden
{
private:
    Sward swards[SWARD_NUM_Y][SWARD_NUM_X + 1]; // 所有草地
public:
    Garden();
    void Print(); // 打印花园
    void Update_Swards(); // 更新所有草地

    ...
};
```

Garden类是花园的类，主要管理了游戏地图中的所有草地。此外，还增加了一列草地，是僵尸进入花园前的空地（马路）。

由于草地是固定不变的，所以不用vector存储，而是采用二维数组来存储。

```
class Sward
{
private:
    int loc_x; // 草地原点的实际x坐标
    int loc_y; // 草地原点的实际y坐标
    Plant* plant; // 草地上的植物
    Plant* extra_plant; // 草地上附加的植物
```

```
    vector<Zombie*> zombies; // 草地上的僵尸
    bool select_flag; // 是否被选中
    bool eat_flag; // 是否正在被吃
    bool update_flag; // 是否需要更新草地
    bool bomb_flag; // 是否正在爆炸
    int bomb_timer; // 爆炸计时器
    int bomb_time; // 爆炸时间
public:
    Sward();
    void Locate(int num_x, int num_y); // 花园坐标与实际坐标的转换
    void Print(); // 打印草地上的内容
    void Print_Select(); // 打印选中标志
    void Print_Bomb(); // 打印爆炸特效
    void Print_Plant(); // 打印草地上的植物
    void Print_Extra_Plant(); // 打印草地上的附加植物
    void Print_Zombie(); // 打印草地上的僵尸
    void Add_Plant(Plant* _plant); // 添加植物
    void Delete_Plant(); // 删除植物
    void Add_Extra_Plant(Plant* _extra_plant); // 添加附加植物
    void Delete_Extra_Plant(); // 删除附加植物
    void Add_Zombie(Garden& garden, Zombie* _zombie); // 添加僵尸
    void Delete_Zombie(Garden& garden, Zombie* _zombie); // 删除僵尸

    ...
};
```

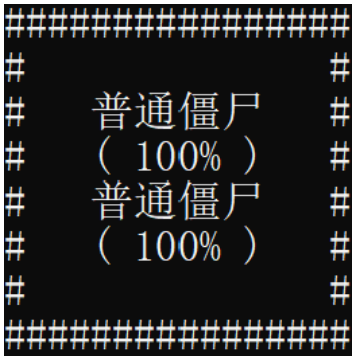
Sward类是每块草地的类，在每块草地上集中管理着该块草地上的植物和僵尸。由于植物和僵尸都是需要进行实时管理的，所以均用指针类型来存储，方便随时的生成、操作和消亡。

规定每块草地上只能有至多一个植物，但可以有多个僵尸。此外，还可以存在一个附加植物（如：南瓜头），用以保护该草地上的植物，也可以单独存在。

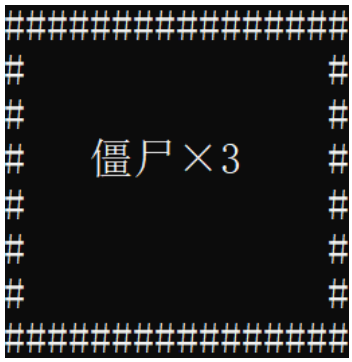
Sward类的另一个作用就是打印草地上的内容，并且有相应的规则：

① 植物 = 0，僵尸 = 1

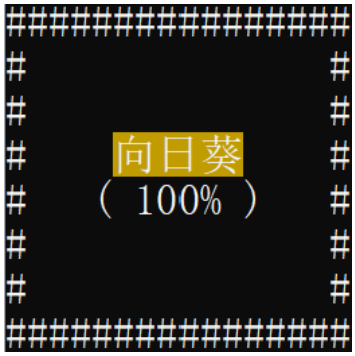
② 植物 = 0， 僵尸 = 2



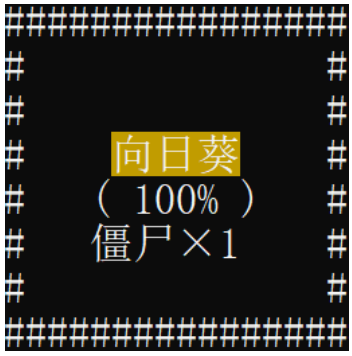
③ 植物 = 0， 僵尸 > 2



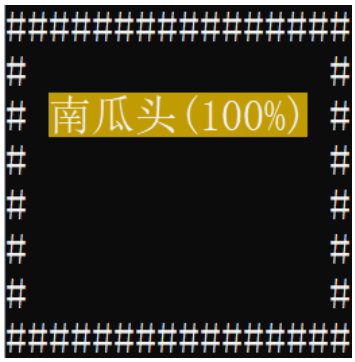
④ 植物 = 1， 僵尸 = 0



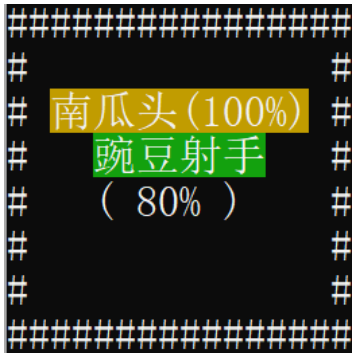
⑤ 植物 = 1， 僵尸 > 0



⑥ 植物 = 0，附加植物 = 1

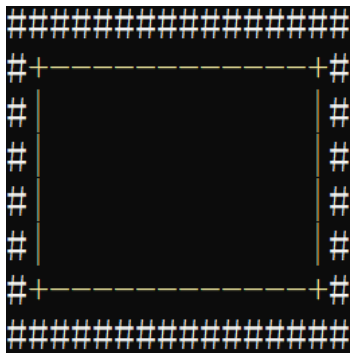


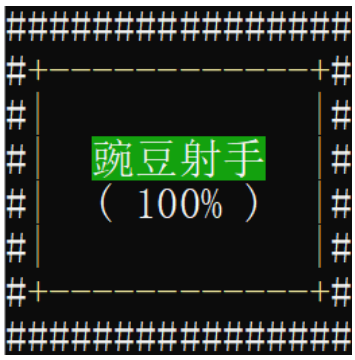
⑦ 植物 = 1，附加植物 = 1



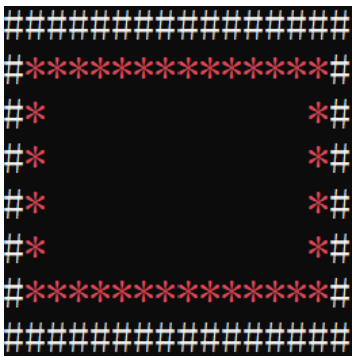
⑧ 植物 = 1，附加植物 = 1，僵尸 > 0

选中标志（无论草地里是否有其他内容）





爆炸特效（无论草地里是否有其他内容）



(5) Plant类

```
class Plant
{
protected:
    string name; // 植物名字
    int hp; // 植物血量
    int hp_left; // 植物剩余血量
    int num_x; // 所在草地的花园x坐标
    int num_y; // 所在草地的花园y坐标
    int timer; // 计时器
    int color; //植物输出颜色
    void Set(int _num_x, int _num_y); // 设置植物的花园坐标
public:
    Plant();
    virtual void Functioning(Game& game){} // 植物功能函数

    ...
};
```

Plant类是植物的基类，存储了植物的一些基本属性，用以在派生类中复用：名字、血量、剩余血量、所在草地的花园坐标、计时器。此外，还用了一个虚函数表示植物的功能函数，植物的每个派生类将会具体实现对应的功能函数。

(6) Zombie类

```
class Zombie
{
protected:
    protected:
    string name; // 僵尸名字
    int hp; // 僵尸血量
    int hp_left; // 僵尸剩余血量
    int attack; // 僵尸攻击力
    int move_speed; // 僵尸移动速度
    int timer; // 计时器
    int kill_score; // 击杀僵尸的得分
    int loc_x; // 僵尸的实际x坐标
    int loc_y; // 僵尸的实际y坐标
    int num_x; // 僵尸的花园x坐标
    int num_y; // 僵尸的花园y坐标
    int color; // 僵尸输出的颜色
    bool freezing_flag; // 是否被冰冻
```

```
    int freezing_time; // 被冰冻的时间
    int freezing_timer; // 被冰冻的计时器
    bool extra_flag; // 是否有额外的饰品
public:
    Zombie();
    void Locate(Garden& garden, int index); // 花园坐标和实际坐标的转换
    void Set(Garden& garden, const string& _name, int _hp, int _attack, int
_move_speed, int _kill_score, int _num_x, int _num_y, int _freezing_time);  // 设
置僵尸相关参数
    virtual bool Move(Game& game); // 僵尸移动
    void Eat(Garden& garden); // 僵尸进食

    ...
};
```

Zombie类是僵尸的基类，同时也是普通僵尸，存储了一些僵尸的基本属性，用以在派生类中复用：名字、血量、剩余血量、攻击力、移动速度、击杀僵尸的得分、实际坐标、花园坐标、输出颜色、冰冻信息、饰品信息。此外，还定义了僵尸的一些行为函数：移动和进食。

(7) Pea类

```
class Pea
{
protected:
    string name; // 豌豆名字
    int speed; // 豌豆速度
    int attack; // 豌豆攻击力
    int color; // 豌豆颜色
    int loc_x; // 豌豆实际x坐标
    int loc_y; // 豌豆实际y坐标
    int num_x; // 豌豆花园x坐标
    int num_y; // 豌豆花园y坐标
    int timer; // 计时器
public:
    Pea();
    Pea(int _loc_x, int _loc_y);
    void Locate(); // 实际坐标与花园坐标的转换
    bool Move(Game& game); // 豌豆移动
    virtual void Hit(Game& game); // 豌豆击中僵尸
    void Print(Garden &garden); // 打印豌豆

    ...
};
```

Pea类是豌豆的基类，同时也是普通豌豆，存储了一些豌豆的基本属性，用以在派生类中复用：名字、速度、攻击力、攻击力、颜色、实际坐标、花园坐标、计时器。此外，还定义了豌豆的一些行为函数：移动、击中僵尸和打印。当击中僵尸或超出边界后自动销毁。

3、植物的派生类

各类植物（和对应商品）的属性如下：

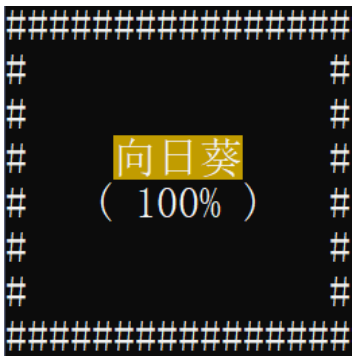
名字	血量	价格	冷却时间(s)	功能速度(s/次)
向日葵	100	50	5	3
豌豆射手	100	100	7.5	3
双发射手	100	200	7.5	3
寒冰射手	100	175	7.5	3
坚果墙	300	50	10	\
高坚果	600	125	12.5	\
窝瓜	100	50	12.5	0.5（遇到僵尸后）
樱桃炸弹	100	150	12.5	1
大蒜	125	50	7.5	\
南瓜头	300	125	12.5	\

（1）向日葵

```
class Sunflower :public Plant
{
private:
    int sunshine_speed; // 阳光产生速度
    int sunshine_produce; // 单次阳光的产生量
public:
    Sunflower() { name = "向日葵"; hp = hp_left = 100; sunshine_speed = 30; color
= SUNFLOWER_COLOR, sunshine_produce = 25; }
    void Functioning(Game& game); // 产生阳光
};
```

向日葵的功能函数是产生阳光，功能是每隔sunshine_speed的时间，产生sunshine_produce数量的阳光。

向日葵效果如下：

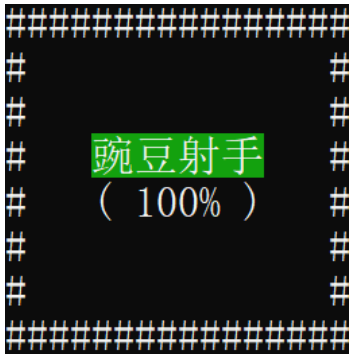


(2) 豌豆射手

```
class Peashooter :public Plant
{
private:
    int pea_speed; // 发射豌豆的速度
public:
    Peashooter() { name = "豌豆射手"; hp = hp_left = 100; color =
PEASHOOTER_COLOR, pea_speed = 30; }
    void Functioning(Game& game); // 发射豌豆
    bool Check_Zombie(Garden& garden); // 判断该行是否有僵尸
};
```

豌豆射手的功能函数是发射豌豆，功能是当所在行有僵尸进入花园时（在马路时不算），每隔pea_speed的时间发射一个豌豆。

豌豆射手效果如下：

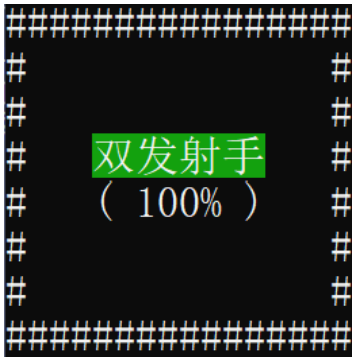


(3) 双发射手

```
class Repeater :public Plant
{
private:
    int pea_speed; // 发射豌豆的速度
    bool shoot_flag; // 是否需要发射第二颗豌豆
public:
    Repeater() { name = "双发射手"; hp = hp_left = 100; color = REPEATER_COLOR;
pea_speed = 30; shoot_flag = false; }
    bool Functioning(Game& game);
    bool Check_Zombie(Garden& garden); // 判断该行是否有僵尸
};
```

双发射手和豌豆射手功能大致相似，但是每次会发射两颗豌豆，对僵尸造成连续的打击。

双发射手效果如下：

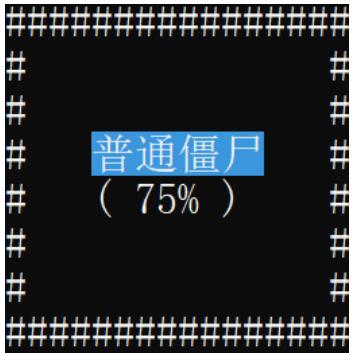


(4) 寒冰射手

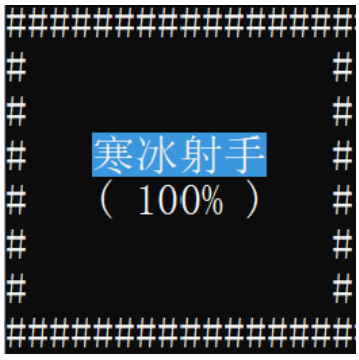
```
class Snowpea :public Plant
{
private:
    int pea_speed; // 发射豌豆的速度
public:
    Snowpea() { name = "寒冰射手"; hp = hp_left = 100; color = SNOWPEA_COLOR;
pea_speed = 30; }
    bool Functioning(Game& game);
    bool Check_Zombie(Garden& garden); // 判断该行是否有僵尸
};
```

寒冰射手与豌豆射手的功能也大致相似，但射出的豌豆是寒冰豌豆，攻击力相同，但会对击中的僵尸产生冰冻效果，使其移动速度减半，进食速度减半（所有操作速度都减半）。

被寒冰射手击中的僵尸会有冰冻特效：



寒冰射手效果如下：



(5) 坚果墙

```
class Wallnut :public Plant
{
private:
public:
    Wallnut() { name = "坚果墙"; hp = hp_left = 300; color = WALLNUT_COLOR;}
    bool Functioning(Game& game);
};
```

坚果墙的功能是阻挡僵尸的前进，有较高的生命值，可以保护后面的植物，但无法阻挡有杆的撑杆僵尸。

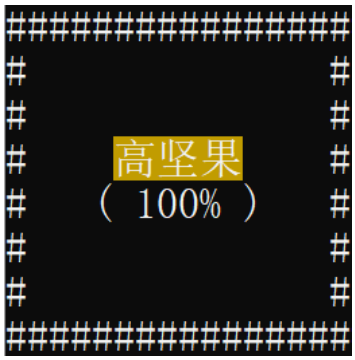
坚果墙效果如下：

(6) 高坚果

```
class Tallnut :public Plant
{
private:
public:
    Tallnut() { name = "高坚果"; hp = hp_left = 600; color = TALLNUT_COLOR;}
    bool Functioning(Game& game);
};
```

高坚果和坚果墙基本一致，拥有坚果墙两倍的生命值。此外，是唯一可以阻挡有杆的撑杆僵尸的植物。

高坚果效果如下：



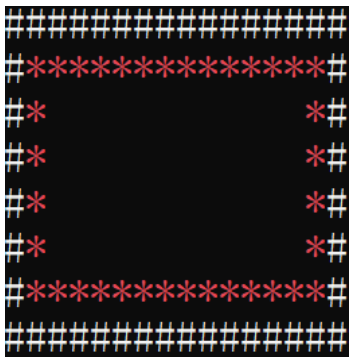
(7) 窝瓜

```
class Squash :public Plant
{
private:
    int bomb_speed; // 从种植到爆炸的时间
public:
    Squash() { name = "窝瓜"; hp = hp_left = 100; color = SQUASH_COLOR;
bomb_speed = 5; }
    bool Functioning(Game& game);
};
```

窝瓜的功能是当所在草地上有僵尸时，经过0.5秒后，消灭出现在所在草地上的所有僵尸，同时自己也会被销毁。

窝瓜效果如下：

窝瓜爆炸效果如下：

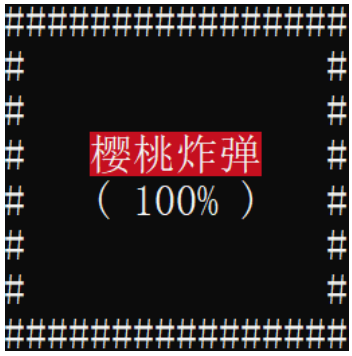


(8) 樱桃炸弹

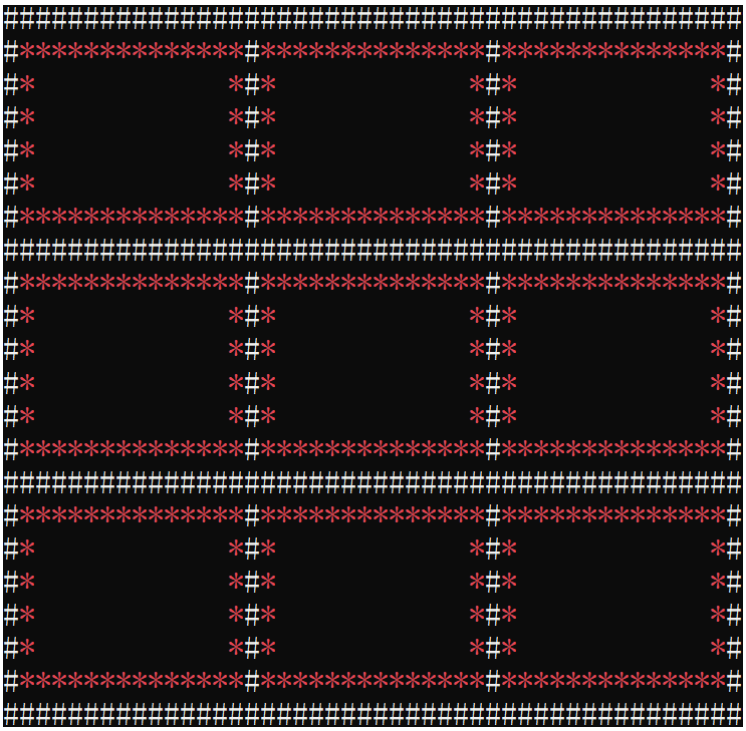
```
class Cherrybomb :public Plant
{
private:
    int bomb_speed; // 从种植到爆炸的时间
public:
    Cherrybomb() { name = "樱桃炸弹"; hp = hp_left = 100; color =
CHERRYBOMB_COLOR; bomb_speed = 10; }
    bool Functioning(Game& game);
};
```

樱桃炸弹的功能是炸毁以自己为中心的3×3的草地内所有的僵尸，种植一秒后即爆炸，无需僵尸激活。

樱桃炸弹效果如下：



樱桃炸弹爆炸效果如下：

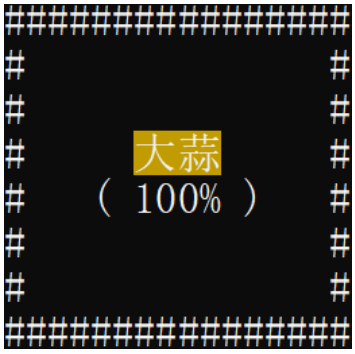


(9) 大蒜

```
class Garlic :public Plant
{
private:
public:
    Garlic() { name = "大蒜"; hp = hp_left = 125; color = GARLIC_COLOR;}
    bool Functioning(Game& game);
};
```

大蒜的功能是使出现在所在草地的僵尸随机移向上下两行中的某一行的同一列，但同时大蒜会先被该僵尸攻击一次。

大蒜效果如下：

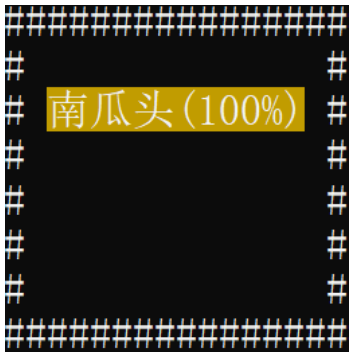


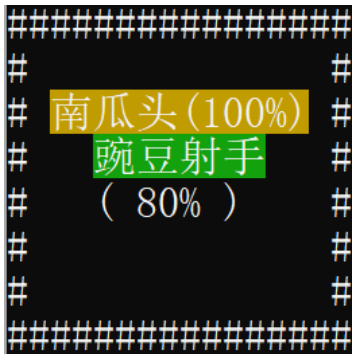
（10）南瓜头

```
class Pumpkin :public Plant
{
private:
public:
    Pumpkin() { name = "南瓜头"; hp = hp_left = 300; color = PUMPKIN_COLOR;}
    bool Functioning(Game& game);
};
```

南瓜头的功能是保护所在草地的植物，拥有较高的生命值，当所在草地出现僵尸时，南瓜头会优先于植物收到僵尸的攻击。也自己也可以单独存在。同一个草地上植物和南瓜头种植的顺序不限。

南瓜头效果如下：





4、僵尸的派生类

各类僵尸的属性如下：

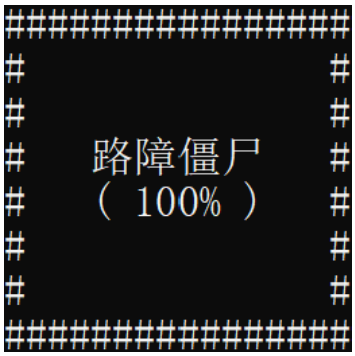
名字	血量	攻击力	移动速度	击杀得分
普通僵尸	100	25	20	10
路障僵尸	150	25	20	15
读报僵尸	125 / 75	25	20 / 40	20
撑杆僵尸	125	25	20	20
小丑僵尸	100	25	20	15
投石僵尸	250	20	20	40

(1) 路障僵尸

```
class Coneheadzombie :public Zombie
{
private:
public:
    Coneheadzombie() :Zombie() {}
    bool Move(Game& game); // 僵尸移动
};
```

路障僵尸与普通僵尸的属性基本一致，但拥有更高的生命值。

路障僵尸效果如下：



(2) 读报僵尸

```
class Newspaperzombie :public Zombie
{
private:
    bool newspaper_flag; // 是否有报纸
public:
    Newspaperzombie() :Zombie() { newspaper_flag = true; }
    bool Move(Game& game); // 僵尸移动
};
```

读报僵尸在有报纸时与普通僵尸基本一致，但当报纸被植物击落后，会转化为愤怒状态，移动速度，进食速度等所有操作速度会变为正常情况的两倍，同时颜色会变为红色。

读报僵尸（有报纸时）效果如下：

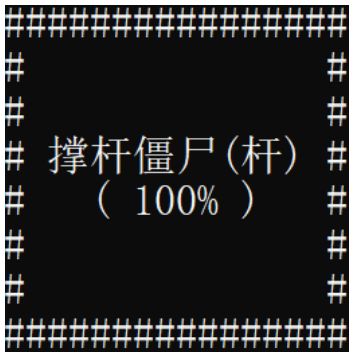
读报僵尸（无报纸时）效果如下：

(3) 撑杆僵尸

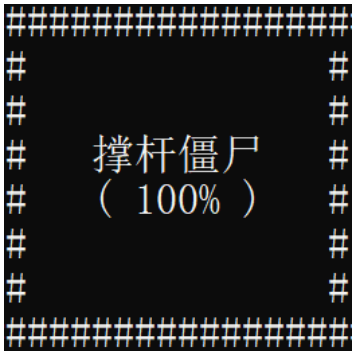
```
class Polevaultingzombie :public Zombie
{
private:
public:
    Polevaultingzombie() :Zombie() {}
    bool Move(Game& game); // 僵尸移动
};
```

撑杆僵尸会越过遇到的第一个植物（除高坚果），然后进行与普通僵尸相同的操作。

撑杆僵尸（有杆）效果如下：



撑杆僵尸（无杆）效果如下：



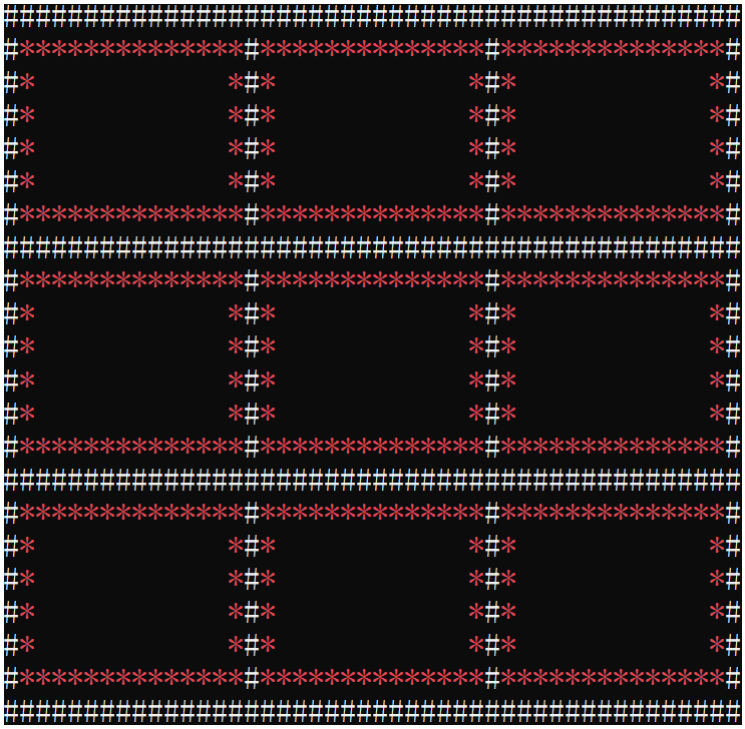
(4) 小丑僵尸

```
class Jackintheboxzombie :public Zombie
{
private:
    bool bomb_flag; // 是否会爆炸
public:
    Jackintheboxzombie() :Zombie() {}
    bool Move(Game& game); // 僵尸移动
};
```

小丑僵尸将会在行进的过程中随机爆炸（爆炸概率随行进的距离增大而增大），炸毁以自己为中心的3×3的草地中所有的植物，自己也会被销毁。

小丑僵尸效果如下：

小丑僵尸爆炸效果如下：



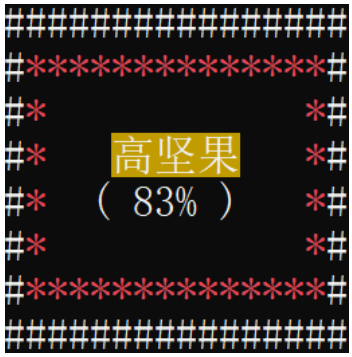
(5) 投石僵尸

```
class Catapultzombie :public Zombie
{
private:
    int stone;
    int stone_left;
public:
    Catapultzombie() :Zombie() { stone = stone_left = 10; }
    bool Move(Game& game); // 僵尸移动
    int Check_Plant(Garden& garden); // 判断该行是否有植物
};
```

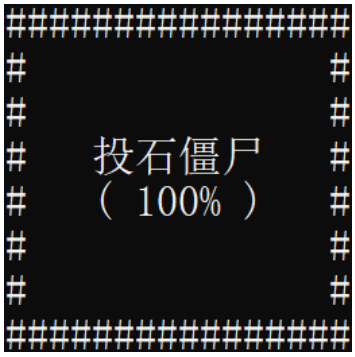
投石僵尸一开始会前进两个草地，若这两个草地上有植物，则会被直接碾压。然后会向所在行的最后一个植物投射石头来攻击，直至存有的十个石头被投完或所在行的所有植物被摧毁完，便开始前进。若所在行没有植物，则会一直前进，当所在行存在植物后，又会停下来，开始投射石头（还有剩余石头的话）。在前进过程中，所遇到的植物会被直接碾压。

投石僵尸（有石头）效果如下：

石头投射植物效果如下：



投石僵尸（无石头）效果如下：



5、豌豆的派生类

（1）寒冰豌豆

```
class SnowPea :public Pea
{
private:
public:
    SnowPea();
    SnowPea(int _loc_x, int _loc_y) :Pea(_loc_x, _loc_y) { color =
PEASNOW_COLOR; };
    void Hit(Game& game); // 豌豆击中僵尸(冰冻)
};
```

寒冰豌豆和普通豌豆的属性基本一致，但颜色是浅蓝色，当击中僵尸后，会有额外的冰冻效果，使僵尸的移动速度，进食速度（所有操作的速度）减半。

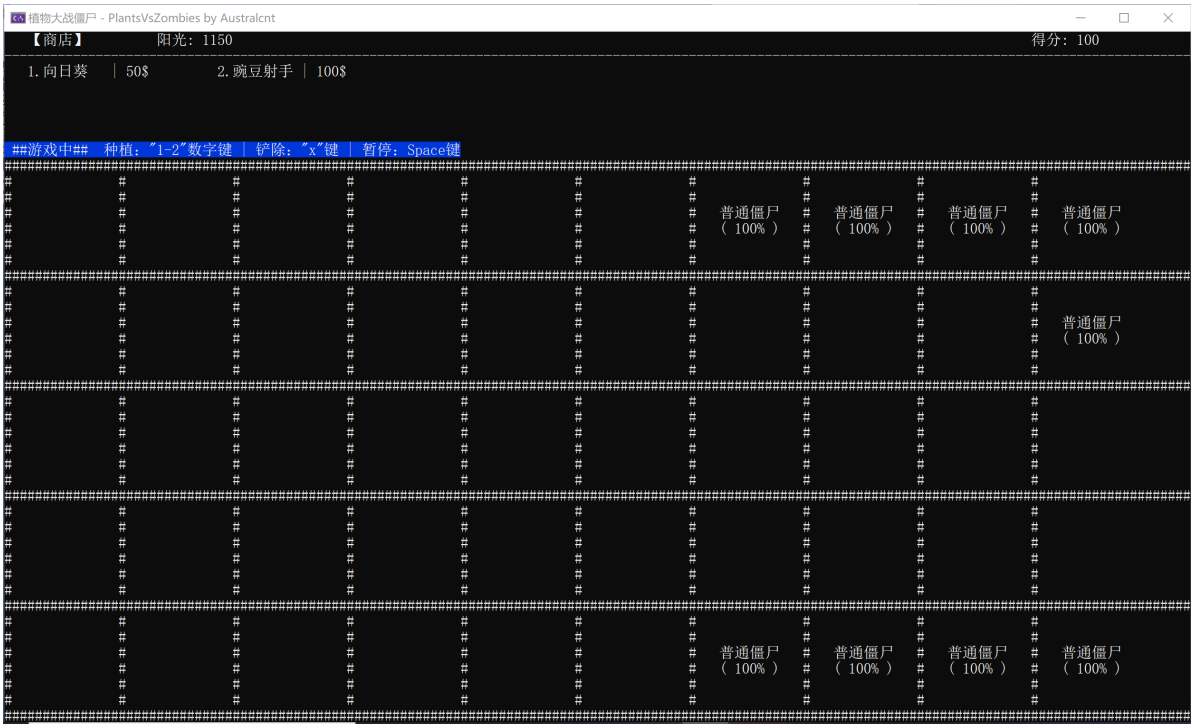
三、游戏效果

1、操作方法

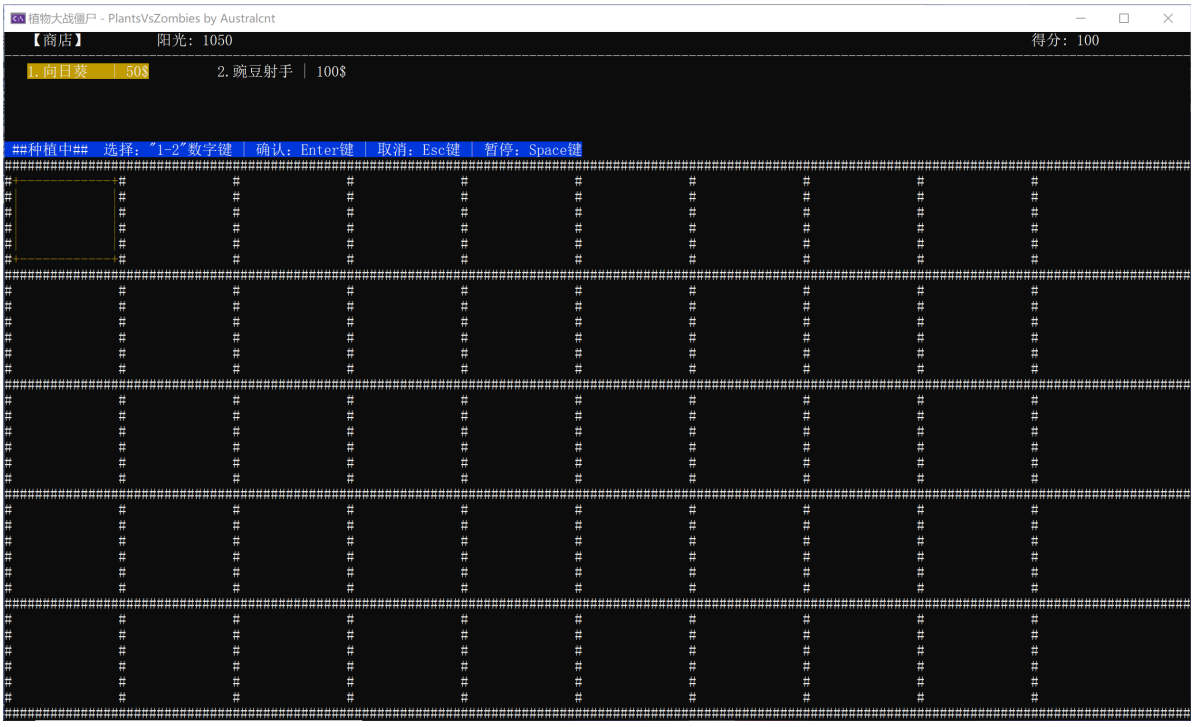
- 移动选中框：方向键
- 购买植物：“1-9”数字键，"a"字母键（不区分大小写）
- 移除植物：“x”字母键（不区分大小写）
- 确认购买/移除：Enter键
- 取消：Esc键
- 暂停：Space键

2、游戏效果

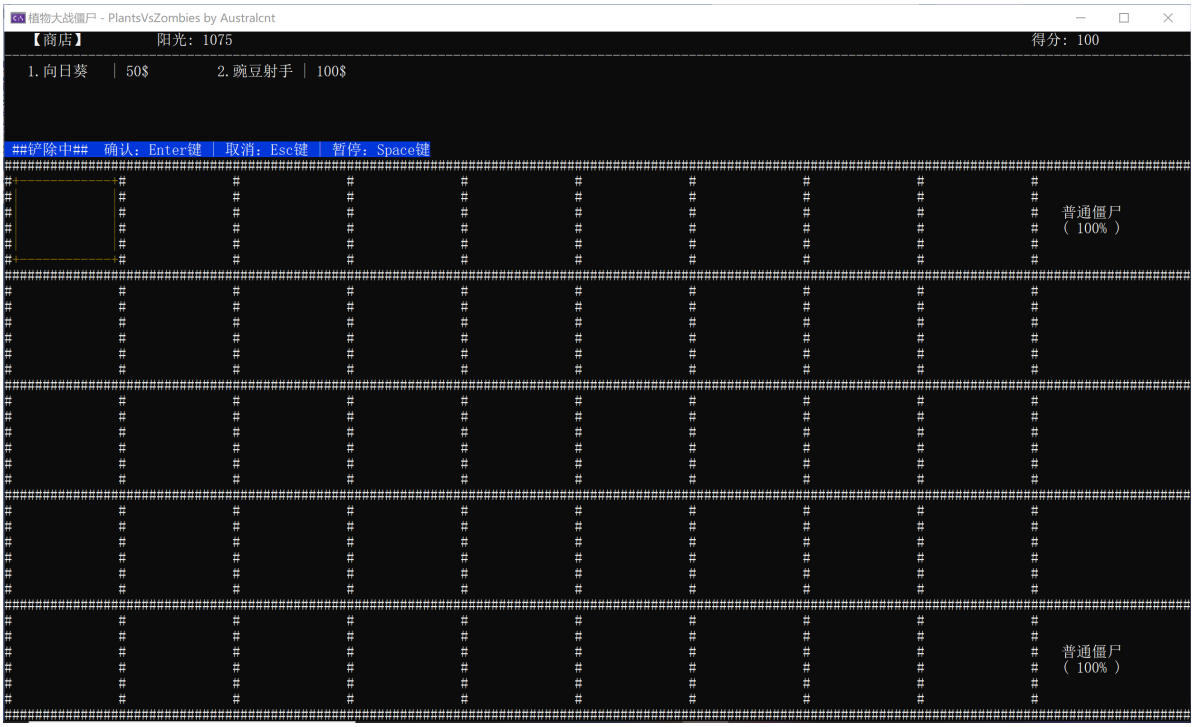
① 正常运行



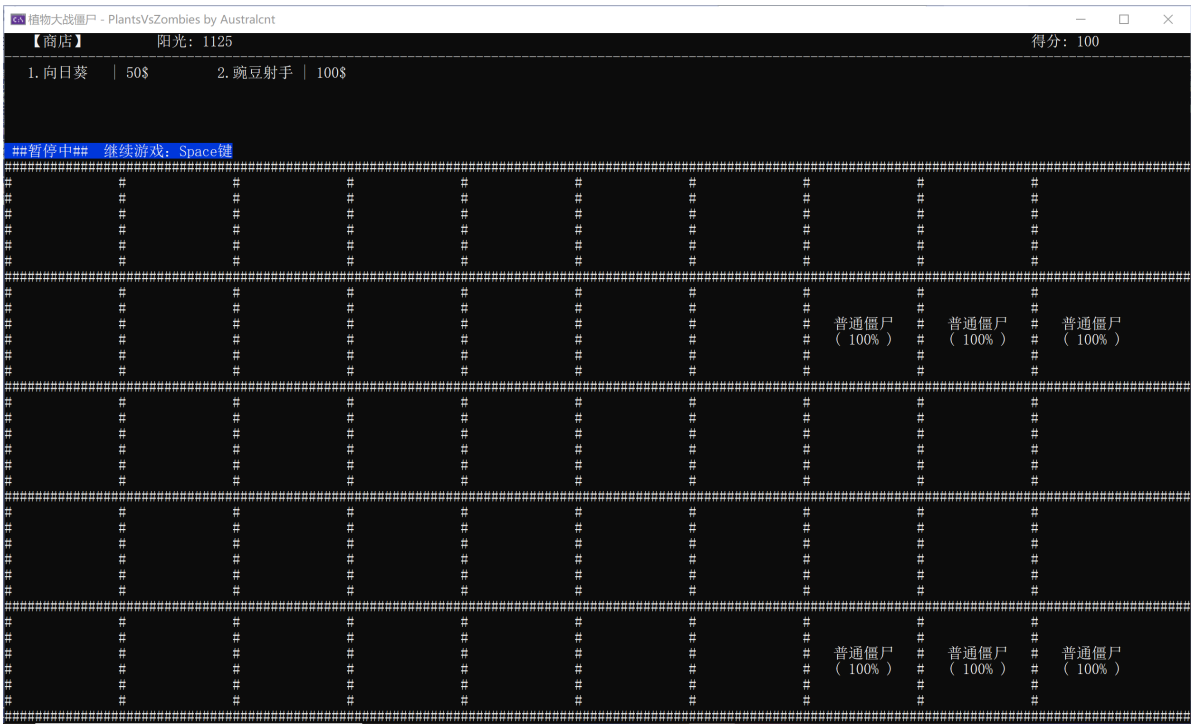
② 购买植物



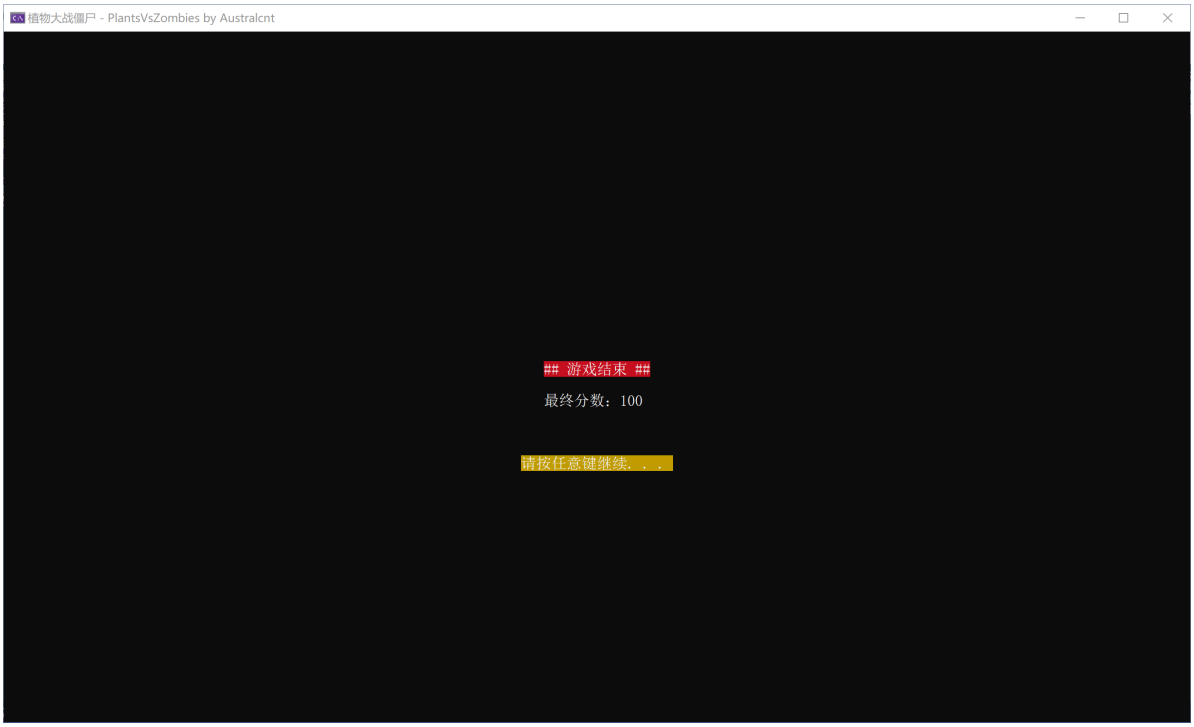
③ 移除植物


植物大战僵尸 - PlantsVsZombies by Australcnt
【商店】 阳光：1075 得分：100
1.向日葵 | 50$ 2.豌豆射手 | 100$
##铲除中## 确认：Enter键 | 取消：Esc键 | 暂停：Space键
普通僵尸
（100%）
普通僵尸
（100%）

④ 暂停游戏


植物大战僵尸 - PlantsVsZombies by Australcnt
【商店】 阳光：1125 得分：100
1.向日葵 | 50$ 2.豌豆射手 | 100$
##暂停中## 继续游戏：Space键
普通僵尸
（100%）
普通僵尸
（100%）
普通僵尸
（100%）
普通僵尸
（100%）
普通僵尸
（100%）
普通僵尸
（100%）

⑤ 游戏结束



四、遇到的问题与解决方案

1、僵尸和豌豆有“残影”

解决方案：原因是未及时清除上一帧留下的画面，所以在打印当前僵尸或豌豆是时候，需要用空格来清除上一帧留下的画面，然后再打印当前画面，并且重新打印上一帧画面所在草地的内容。

2、画面闪烁严重

解决方案：采用人为规定时钟周期的方式，游戏的一次主循环循环即可定义为一个时钟周期，在每个周期对需要更新的内容进行重新打印。本游戏中定义的时钟周期为100ms，所以游戏中其他所有与时间有关的变量均是实际时间的十倍大，表示时间周期的数目。因此，我们可以基于此，在每个时钟周期对需要更新的内容进行记录，并在时钟周期的末尾统一进行更新打印，这样就可表面画面闪烁严重的问题。

```
class Game
{
private:
    ...
    int score; // 分数
    bool score_update_flag; // 是否需要更新分数
    ...
public:
    ...
    void Update_Score(); // 更新分数
    ...
};
```

```
class Store
{
private:
```

```
    int sunshine; // 阳光数
    int sunshine_speed; // 自然阳光产生速度
    int sunshine_produce; // 单次产生的自然阳光数
    bool sunshine_update_flag; // 是否需要更新阳光数
    ...
public:
    ...
    void Sunshine_Produce(Game& game); // 产生自然阳光
    void Update_Sunshine(Game& game); // 更新阳光数
    ...
};
```

```
class Sward
{
private:
    ...
    bool update_flag; // 是否需要更新草地
public:
    ...
    void Print(); // 打印草地上的内容
    void Print_Select(); // 打印选中标志
    void Print_Plant(); // 打印草地上的植物
    void Print_Zombie(); // 打印草地上的僵尸
    ...
};

class Garden
{
private:
    Sward swards[SWARD_NUM_Y][SWARD_NUM_X + 1]; // 所有草地
public:
    ...
    void Update_Swards(); // 更新所有草地
    ...
};
```

时钟周期末统一更新打印：

```
void Game::Game_Start() // 游戏主要逻辑
{
    ...
    while(1)
    {
        ...
        if (score_update_flag == true)
            Update_Score();
        if (store.sunshine_update_flag == true)
            store.Update_Sunshine(*this);
        garden.Update_Swards();
        ...

        Sleep(SLEEP_TIME);
    }
}
```