第一次课程设计报告

植物大战僵尸 (控制台版) —— 基础版

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：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

（3）僵尸

- 普通僵尸

（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

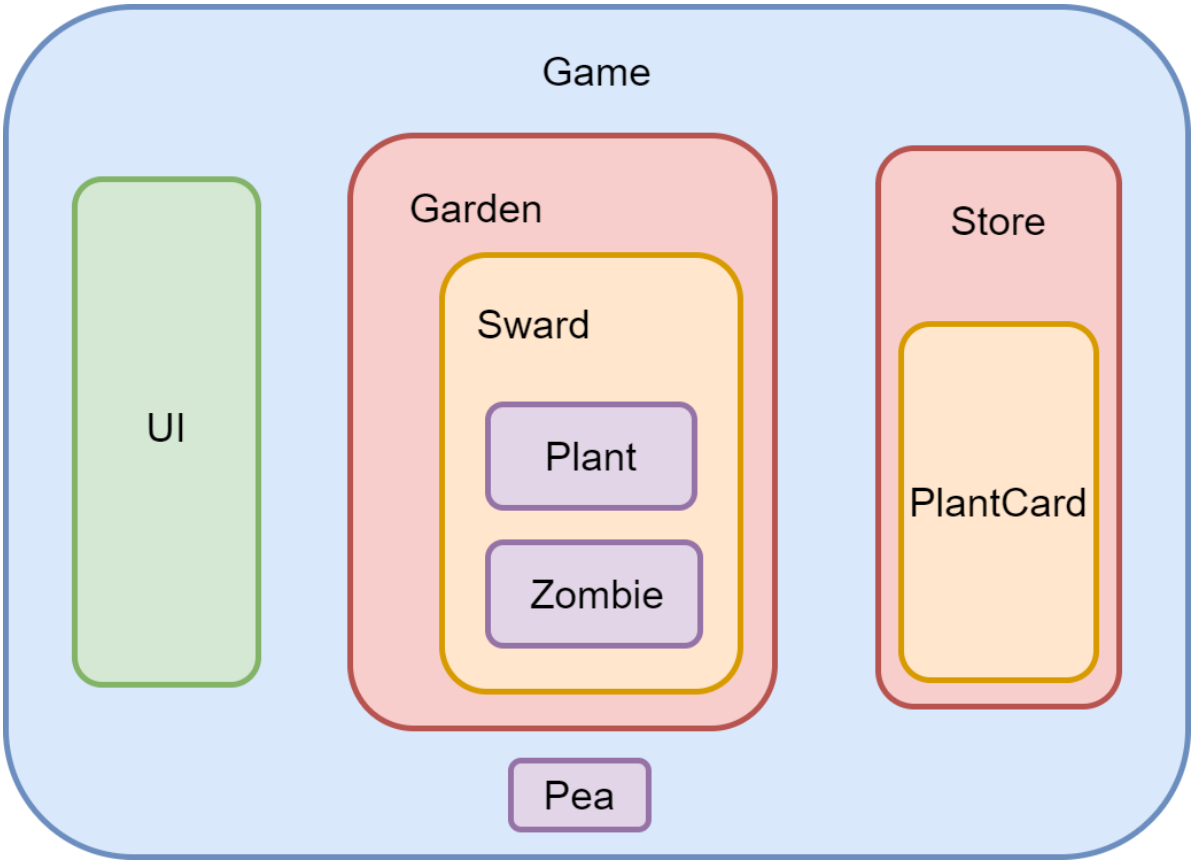
  ⑤ 僵尸

  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
    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;
    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。

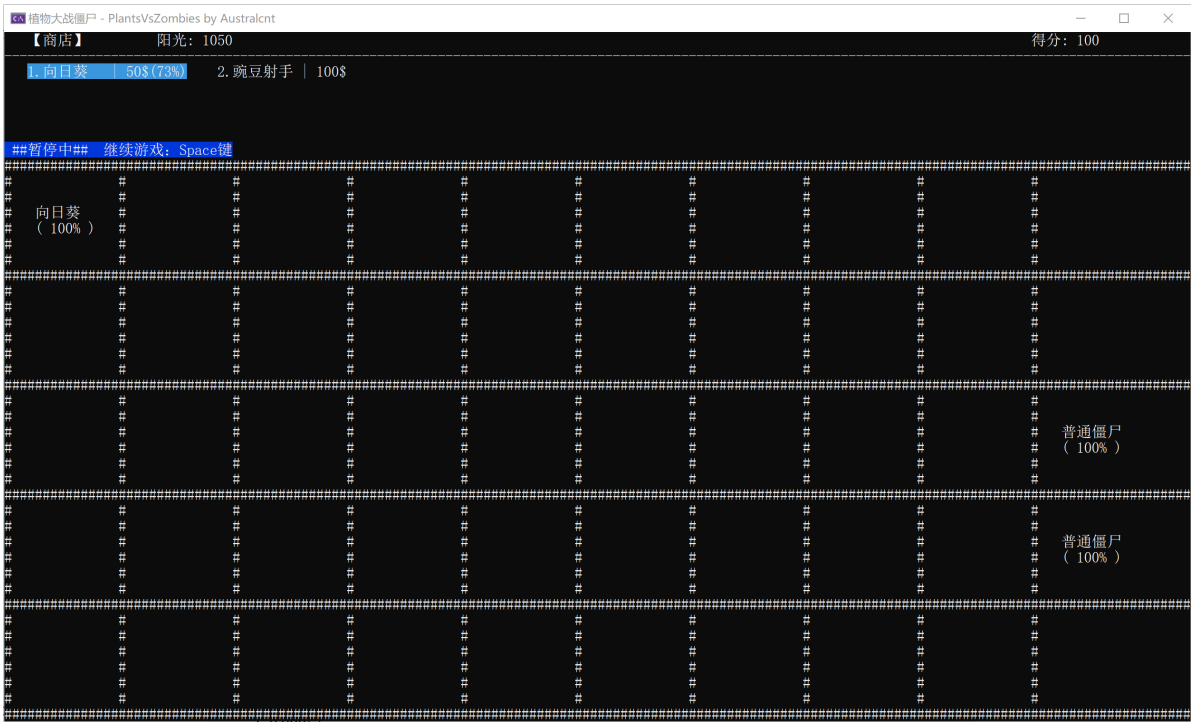
由于商品种类是固定不变的，所以不用vector存储，而是采用一维数组进行存储。

```
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
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
    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（植物闪烁）
    bool update_flag; // 是否需要更新
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_Plant(); // 打印草地上的植物
    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_Zombie(Garden& garden, Zombie* _zombie); // 添加僵尸
    void Delete_Zombie(Garden& garden, Zombie* _zombie); // 删除僵尸

    ...
};
```

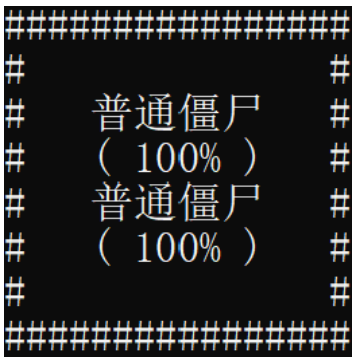
Sward类是每块草地的类，在每块草地上集中管理着该块草地上的植物和僵尸。由于植物和僵尸都是需要进行实时管理的，所以均用指针类型来存储，方便随时的生成、操作和消亡。

规定每块草地上只能有至多一个植物，但可以有多个僵尸。

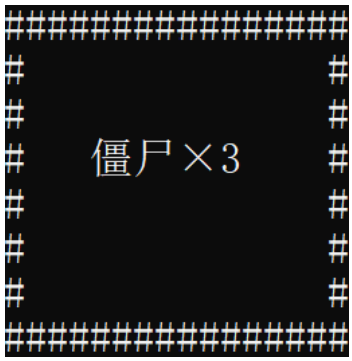
Sward类的另一个作用就是打印草地上的内容，并且有相应的规则：

① 植物 = 0，僵尸 = 1

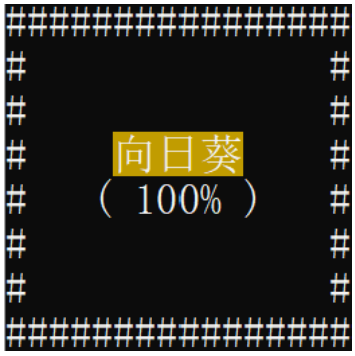
② 植物 = 0， 僵尸 = 2



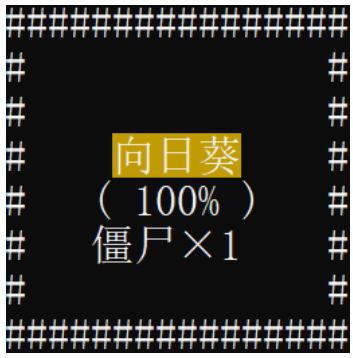
③ 植物 = 0， 僵尸 > 2



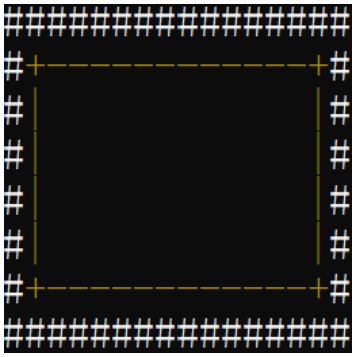
④ 植物 = 1， 僵尸 = 0

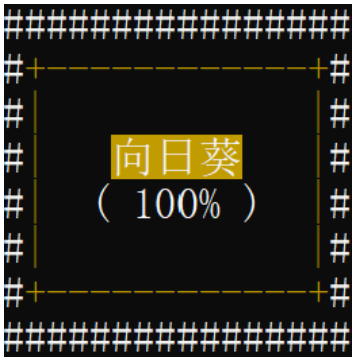


⑤ 植物 = 1， 僵尸 > 0



选中标志（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:
    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
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 = "普通僵尸", int _hp = 100, int
_attack = 25, int _move_speed = 50, int _kill_score = 10, int _num_x =
SWARD_NUM_X, int _num_y = rand() % SWARD_NUM_Y);  // 设置僵尸相关参数
    bool Move(Game& game); // 僵尸移动
    void Eat(Garden& garden); // 僵尸进食

    ...
};
```

Zombie是僵尸的基类，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。此外，还定义了僵尸的一些行为函数：移动和进食。

3、植物的派生类

各种植物（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)
豌豆射手	100	100	7.5	2
向日葵	100	50	5	3

（1）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。

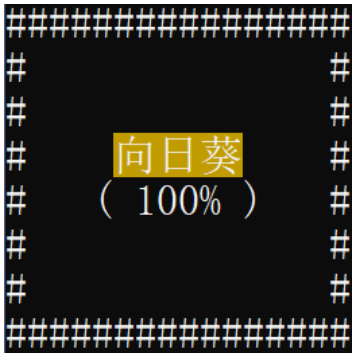
打印豌豆射手时会有对应颜色：

（2）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。

打印向日葵时会有对应颜色：



4、僵尸的派生类

暂无

5、豌豆的派生类

暂无

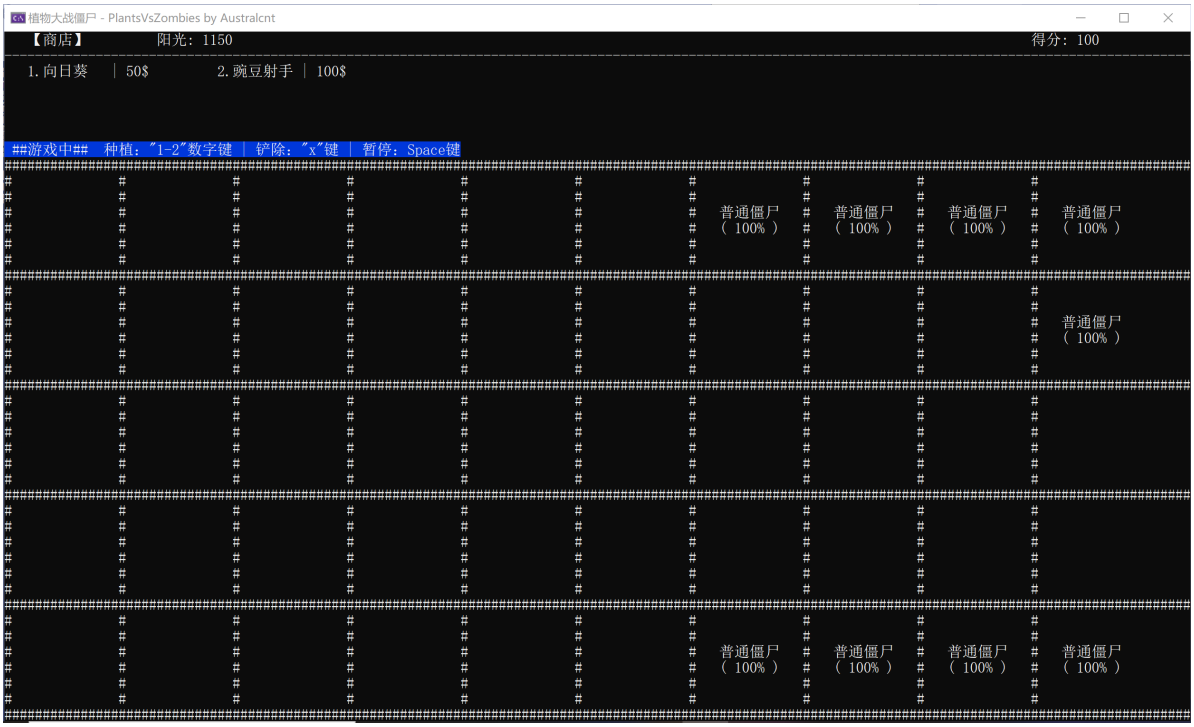
三、游戏效果

1、操作方法

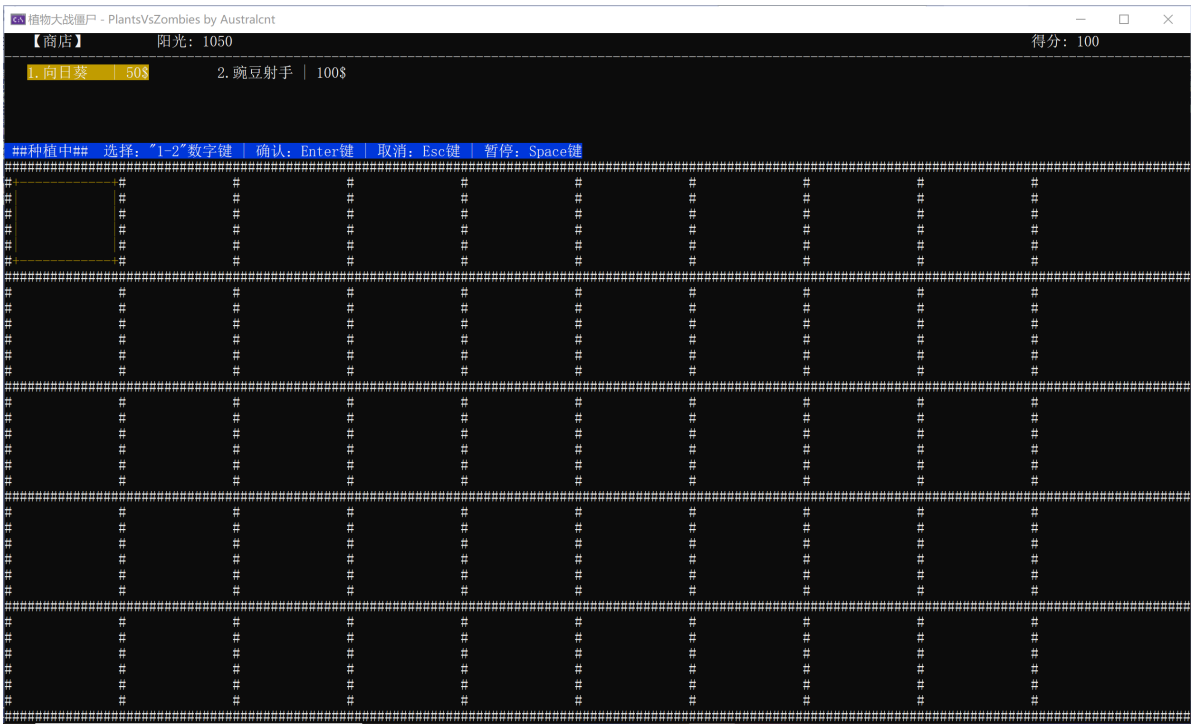
- 移动选中框：方向键
- 购买植物：“1-2”数字键
- 移除植物：“x”字母键（不区分大小写）
- 确认购买/移除：Enter键
- 取消：Esc键
- 暂停：Space键

2、游戏效果

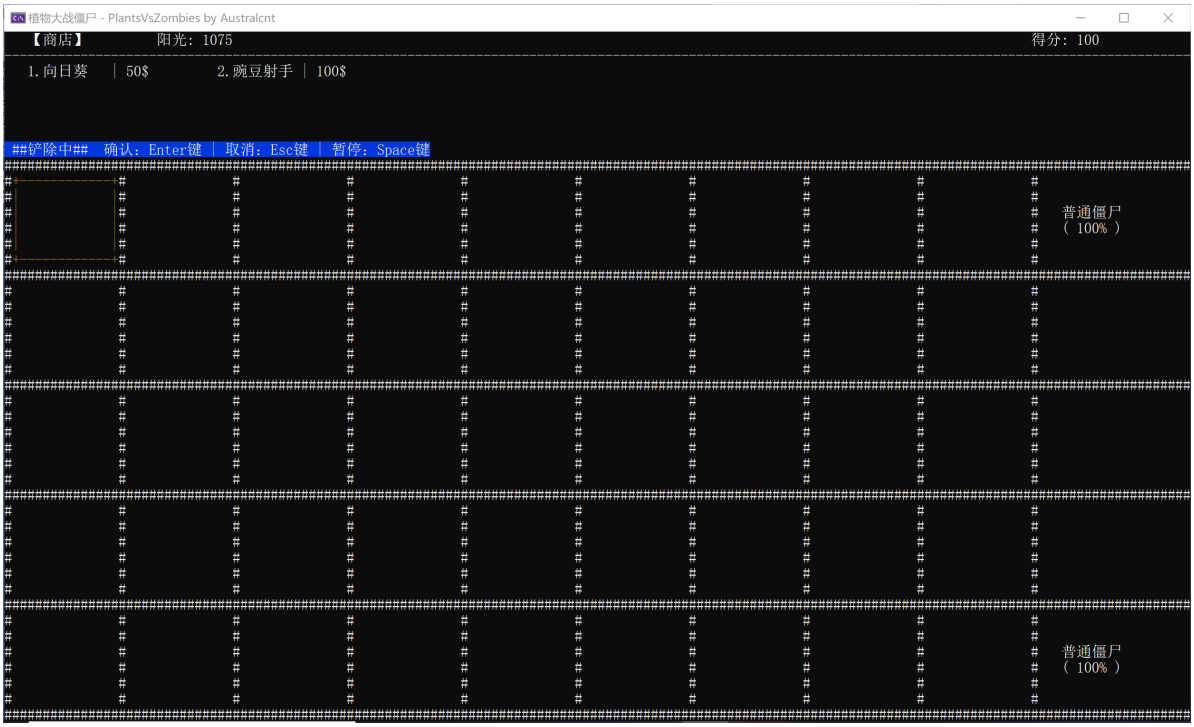
① 正常运行


植物大战僵尸 - PlantsVsZombies by Australcnt
【商店】 阳光：1150 得分：100
1.向日葵 | 50$ 2.豌豆射手 | 100$
##游戏中## 种植："1-2"数字键 | 铲除："x"键 | 暂停：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%)
普通僵尸 (100%)
普通僵尸 (100%)

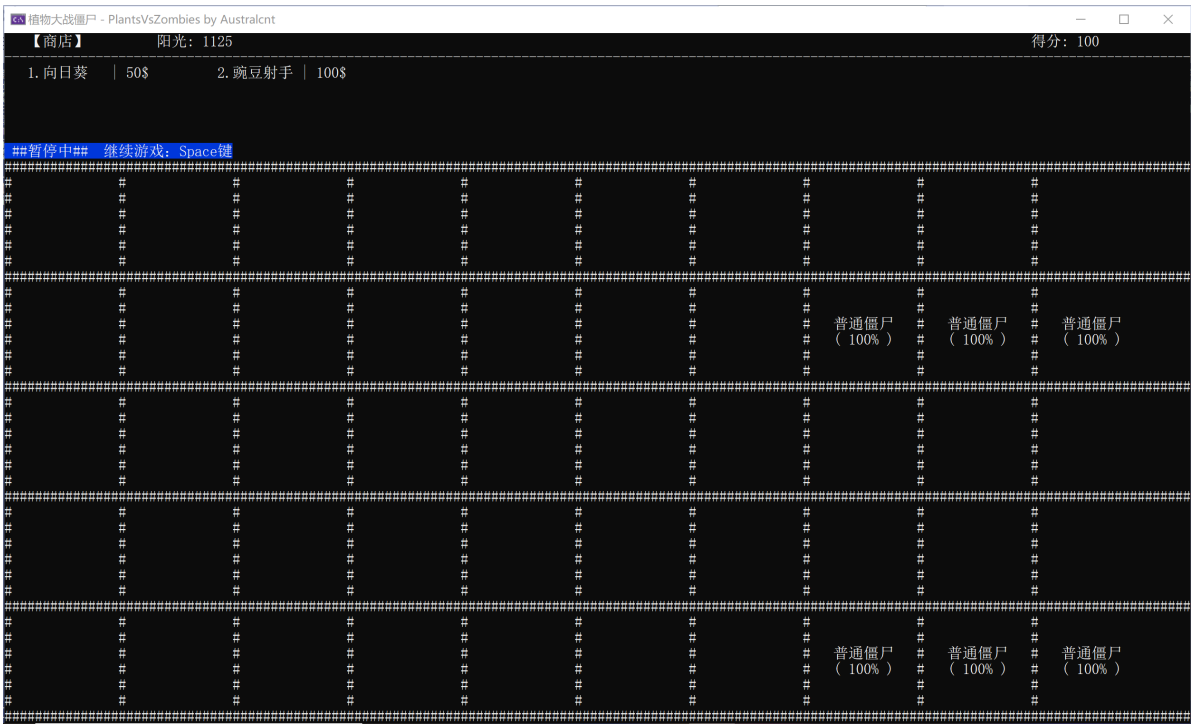
② 购买植物


植物大战僵尸 - PlantsVsZombies by Australcnt
【商店】 阳光：1050 得分：100
1.向日葵 | 50$ 2.豌豆射手 | 100$
##种植中## 选择："1-2"数字键 | 确认：Enter键 | 取消：Esc键 | 暂停：Space键

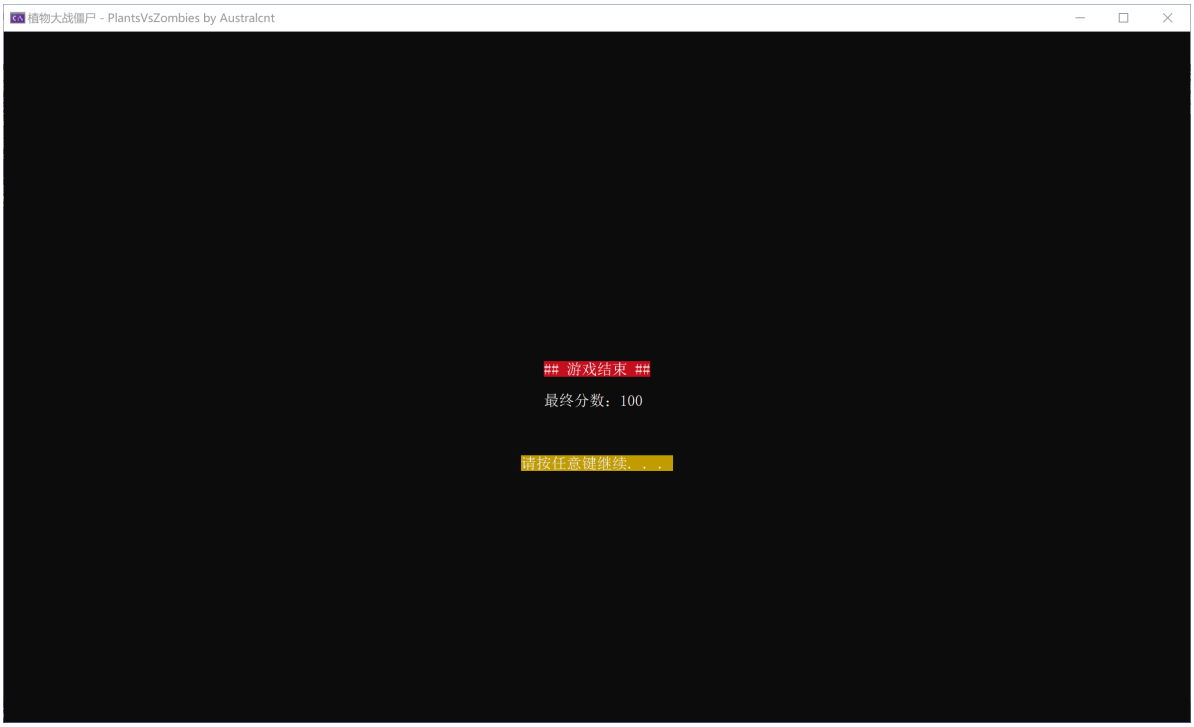
③ 移除植物


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);
    }
}
```