Plants Vs. Zombies Simplified Version

作者：201220120 单博

一、主要内容及实现目标

> 根据知名游戏《植物大战僵尸》，实现一个简化版本。
>
> 第一阶段：基于控制台实现相关内容，采用面向对象的方法进行程序的设计。

第一阶段实现目标

- 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）
- 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，所以没有胜利条件，目标是能够持续抵挡僵尸的进攻，以获得更多的累计积分

基本实现要求

- 庭院布局至少3行7列
- 实现每隔一段时间，系统会产生自然光
- 实现一种僵尸：普通僵尸
- 实现僵尸的三种属性：生命力、攻击力、速度
- 假定每一个地块中只能有一种僵尸
- 实现一种植物：豌豆射手
- 实现植物的三种属性：购买所需阳光数、生命值、攻击力
- 实现植物的购买、地块的选择和植物的种植
- 实现当前局的记分牌

二、实现思路及相关代码

1.相关类设计思路

1.1 植物类Plant

- 属性
    - 生命值
    - 攻击力
    - 攻击速度
    - 阳光价格
    - 坐标x
    - 坐标y
    - 创建时间

    - 植物类型
- 方法
    - 初始化方法
    - 获取植物类型
    - 设定创建时间
    - 获取创建时间
    - 判定是否攻击
    - 受到伤害减少生命值
    - 获取某一植物类型的阳光价格

相关代码如下：

```
// Plants.h
class Plant {
private:
    int lifepoint,
        attackpoint,
        speed,
        cost;
    int x, y, birth;
    PlantType type;
public:
    Plant();
    Plant(int _x, int _y, PlantType _type);
    PlantType GetType() const;
    void SetBirth(int b);
    int GetBirth() const;
    bool AttackCheck(int tick) const;
    bool ReduceLifepoint(int atk);
    static int GetCost(PlantType _type);
};
```

同时建立植物类型到初始数据的映射：

```
// Plants.h
#include <map>

enum class PlantType {
    _NULL = -1,
    _Peashooter,
    _SunFlower,
};

struct PlantData {
    int lifepoint,
        attackpoint,
        speed,
        cost;
};

static std::map <PlantType, PlantData> plants;
```

在植物类的实现过程中添加对上述映射初始化的相关函数：

```cpp
// Plants.cpp
bool PlantDataInit() {
    // lifepoint; attackpoint; speed; cost;
    plants[PlantType::_NULL] = PlantData{
        0, 0, 0, 0
    };
    plants[PlantType::_Peashooter] = PlantData{
        300, 20, 14, 100
    };
    return true;
}
```

1.2 子弹类Bullet

实现思路大体与植物类相似。

- 属性
    - 攻击力
    - 移动速度
    - 坐标x
    - 坐标y
    - 创建时间
    - 子弹类型
- 方法
    - 初始化方法
    - 获取子弹类型
    - 设定创建时间
    - 获取创建时间
    - 判定是否移动
    - 子弹重叠相加
    - 设定横纵坐标
    - 获取子弹当前攻击力

相关代码如下：

```cpp
// Bullets.h
class Bullet {
private:
    int attackpoint,
        speed;
    int x, y, birth;
    BulletType type;
public:
    Bullet();
    Bullet(int _x, int _y, BulletType _type);
    BulletType GetType() const;
    void SetBirth(int b);
    int GetBirth() const;
    bool MoveCheck(int tick) const;
    void AddBullet(const Bullet& bullet);
    void SetLocation(int _x, int _y);
```

```
    int GetAttackPoint() const;
};
```

建立映射与初始化方法参考植物类，相关代码如下：

```
// Bullets.h
#include <map>

enum class BulletType {
    _NULL = -1,
    _NORMAL,
    _FREEZING,
    _BURNING,
};

struct BulletData {
    int attackpoint,
        speed;
};

static std::map <BulletType, BulletData> bullets;

// Bullets.cpp
bool BulletDataInit() {
    // attackpoint; speed;
    bullets[BulletType::_NULL] = BulletData{
        0, 0
    };
    bullets[BulletType::_NORMAL] = BulletData{
        20, 5
    };
    return true;
}
```

1.3 僵尸类Zombie

实现思路大体与植物类相似。

- 属性
    - 生命值
    - 攻击力
    - 移动速度
    - 攻击速度
    - 坐标x
    - 坐标y
    - 创建时间
    - 僵尸类型
- 方法
    - 初始化方法
    - 获取僵尸类型
    - 设定创建时间
    - 获取创建时间

- 判定是否移动
- 判定是否攻击
- 受到伤害减少生命值
- 设定横纵坐标
- 获取某一僵尸类型的击杀得分
- 获取某一僵尸类型的攻击力

相关代码如下：

```
// Zombies.h
class Zombie {
private:
    int lifepoint,
        attackpoint,
        movespeed,
        attackspeed;
    int x, y, birth;
    ZombieType type;
public:
    Zombie();
    Zombie(int _x, int _y, ZombieType _type);
    ZombieType GetType() const;
    void SetBirth(int b);
    int GetBirth() const;
    bool MoveCheck(int tick) const;
    bool AttackCheck(int tick) const;
    bool ReduceLifepoint(int atk);
    void SetLocation(int _x, int _y);
    static int GetPoint(ZombieType _type);
    static int GetAttackPoint(ZombieType _type);
};
```

建立映射与初始化方法参考植物类，相关代码如下：

```
// Zombies.h
#include <map>

enum class ZombieType {
    _NULL = -1,
    _NORMAL,
};

struct ZombieData {
    int lifepoint,
        attackpoint,
        movespeed,
        attackspeed,
        point;
};

static std::map <ZombieType, ZombieData> zombies;

// Zombies.cpp
```

```
bool ZombieDataInit() {
    // lifepoint; attackpoint; movespeed; attackspeed;
    zombies[ZombieType::_NULL] = ZombieData{
        0, 0, 0, 0, 0
    };
    zombies[ZombieType::_NORMAL] = ZombieData{
        100, 100, 47, 10, 1
    };
    return true;
}
```

1.4 提取主类Object

通过对以上三个类的相关属性方法的分析，可以得到它们的共同属性和方法，将其提取成主类Object，这三个类便作为Object的子类继承属性和方法。

Object的相关方法和属性：

- 属性
  - 生命值
  - 攻击力
  - 移动速度
  - 攻击速度
  - 坐标x
  - 坐标y
  - 创建时间
- 方法
  - 初始化方法
  - 设定创建时间
  - 获取创建时间
  - 判定是否移动
  - 判定是否攻击
  - 设定横纵坐标

注：由于对主类继承成员的调用依旧需要利用using进行显式声明，故仅在此进行提取操作，实际过程中并未进行对于Object类的实现。

1.5 平台类Platform

用于存储某一地块的相关信息。

- 属性
  - 坐标x
  - 坐标y
  - 植物指针
  - 僵尸指针
  - 子弹指针
- 方法
  - 初始化方法
  - 添加植物
  - 添加子弹
  - 添加僵尸
  - 移除植物

- 移除子弹
- 移除僵尸
- 获取此地块植物
- 获取此地块僵尸
- 获取此地块子弹

相关代码如下：

```
// Platform.h
#include "Plants.h"
#include "Zombies.h"
#include "Bullets.h"

class Platform {
private:
    int x, y;
    Plant* plant;
    Zombie* zombie;
    Bullet* bullet;
public:
    Platform();
    Platform(int _x, int _y);
    bool AddPlant(PlantType type);
    bool AddZombie(ZombieType type);
    bool AddZombie(Zombie const* _zombie);
    bool AddBullet(BulletType type);
    bool AddBullet(Bullet const* _bullet);
    void RemovePlant();
    void RemoveZombie();
    void RemoveBullet();
    Plant* GetPlant();
    Zombie* GetZombie();
    Bullet* GetBullet();
};
```

1.6 游戏类Game

储存所有地块信息以及游戏相关数据。

- 属性
  - 平台指针数组
  - 当前阳光
  - 当前得分
  - 计时器
- 方法
  - 初始化方法
  - 打印到屏幕
  - 进行一次操作
  - 判定是否可以购买某一类型植物
  - 购买植物
  - 添加植物
  - 添加僵尸
  - 添加子弹

- 获取当前得分

同时在本头文件中存储了游戏进行时需要的相关参数：

- 行列数
- 一次自然增加的阳光数量
- 自然增加阳光的周期
- 游戏计时周期（防止溢出）
- 僵尸生成周期

相关代码如下：

```
// Game.h
#include "Platform.h"

const int WIDTH = 3, LENGTH = 7;
const int SUNLIGHTINCREMENT = 25;
const int SUNLIGHTPRODUCECYCLE = 15;
const int CYCLE = 1e6;
const int ZOMBIEPRODUCECYCLE = 30;

class Game {
private:
    Platform* platforms[WIDTH][LENGTH];
    int sunlight, point, tick;
public:
    Game();
    ~Game();
    void Display(int x, int y) const;
    bool Run();
    bool PlantAffordable(PlantType type) const;
    void PlantTrade(PlantType type);
    bool AddPlant(int x, int y, PlantType type);
    bool AddZombie(int x, int y, ZombieType type);
    bool AddBullet(int x, int y, BulletType type);
    int GetPoint() const;
};
```

2. 相关函数设计思路

2.1 游戏界面Interface

在此文件中定义的游戏界面相关参数：

- 植物种类计数
- 商店选择判定
- 地块选择判定
- 待选植物位置
- 待选地块坐标
- 游戏结束判定

在此文件中定义的游戏界面相关函数：

- 打印欢迎界面
- 打印结束界面

- 游戏开始
- 打印商店
- 打印游戏界面
- 游戏结束

在此文件中定义的游戏界面相关对象：

- 游戏类对象game
- 线程类对象display

2.2 主函数Main

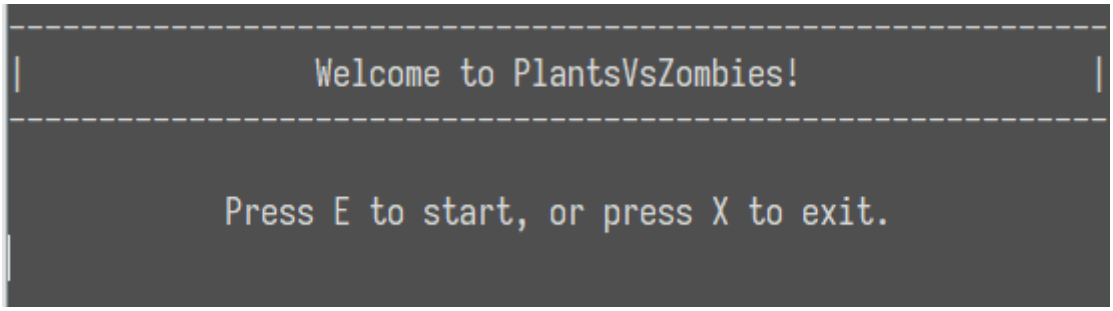
调用类数据的初始化函数，然后调用游戏界面接口，正式开始游戏。

2.3 调用逻辑

1. 主函数调用游戏开始函数；
2. 游戏开始函数内部：
    1. 调用打印欢迎界面界面函数并根据返回值进行决策：
        1. 调用打印结束界面并退出程序；
        2. 流程继续。
    2. 创建打印游戏界面的线程，并将其与主线程分离；
    3. 进入while循环，并根据玩家输入的操作符进行游戏参数的修改；
3. 打印游戏界面函数内部：
    1. 进入while-true循环；
    2. 在while循环中：调用game的进行一次操作方法，根据返回值判断游戏是否结束：
        1. 如果游戏未结束，调用game的打印到屏幕方法以及展示商店函数，停顿100ms；
        2. 如果游戏结束，跳出循环；
    3. 将游戏结束判定设为真，调用游戏结束函数，

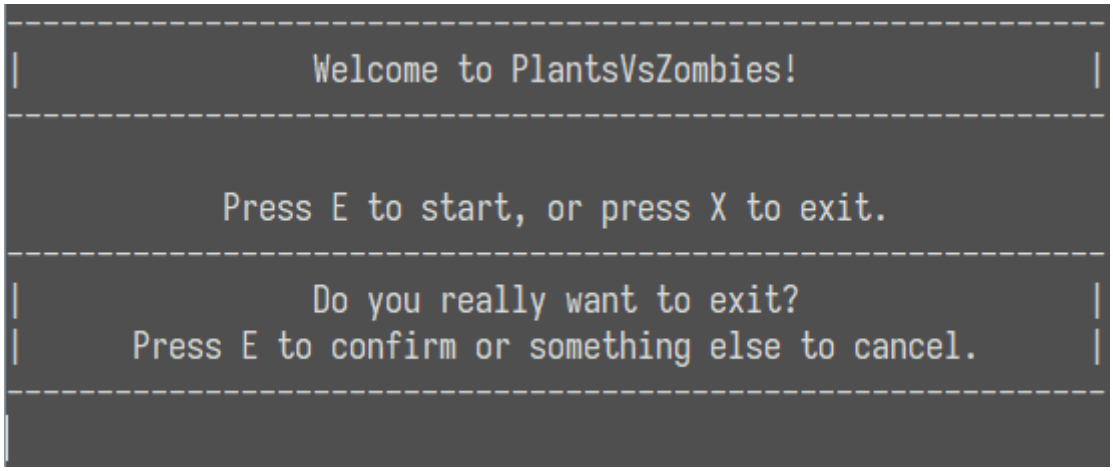
三、程序功能及操作方法

1. 游戏欢迎界面及结束界面

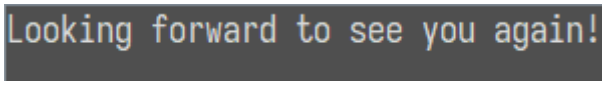


欢迎界面

按下E键进入游戏，按下X键退出游戏。

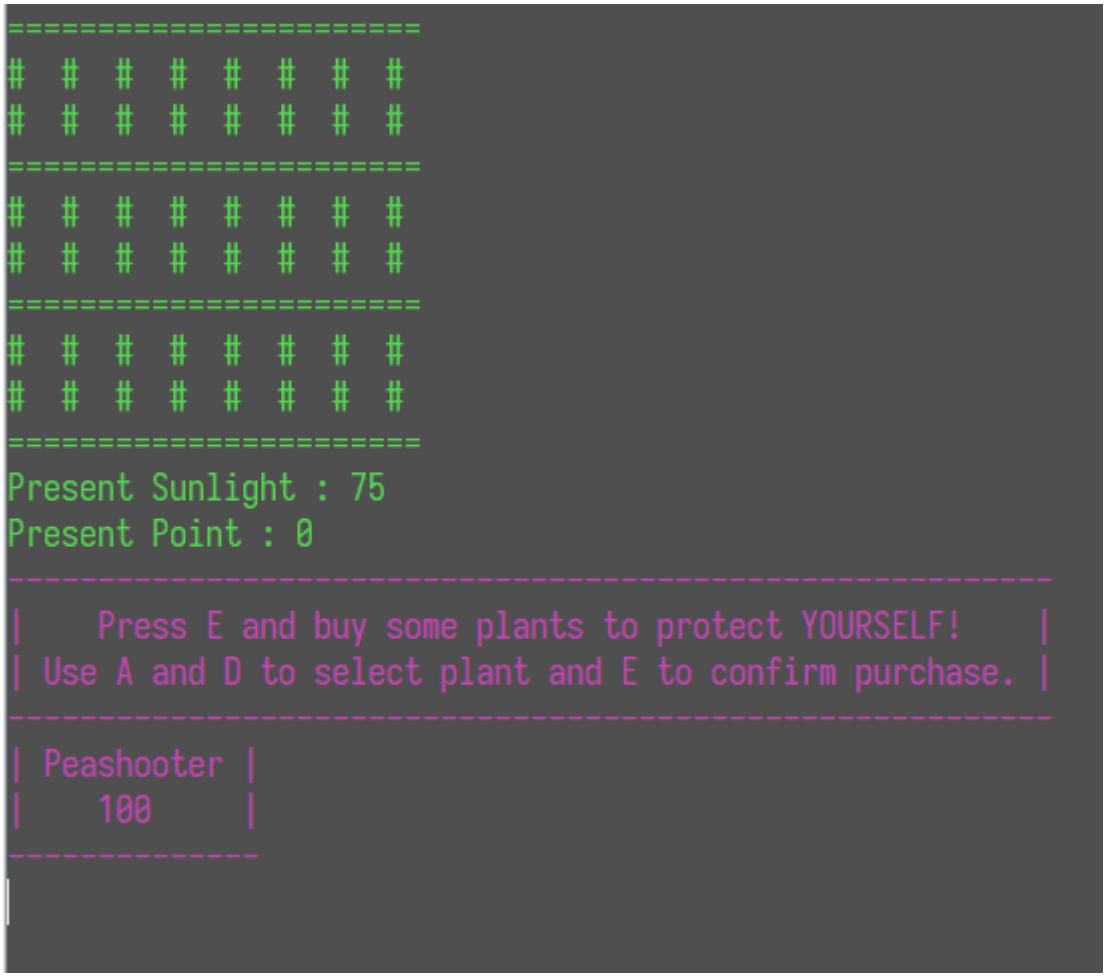


按下X键后



按下E键确定退出后

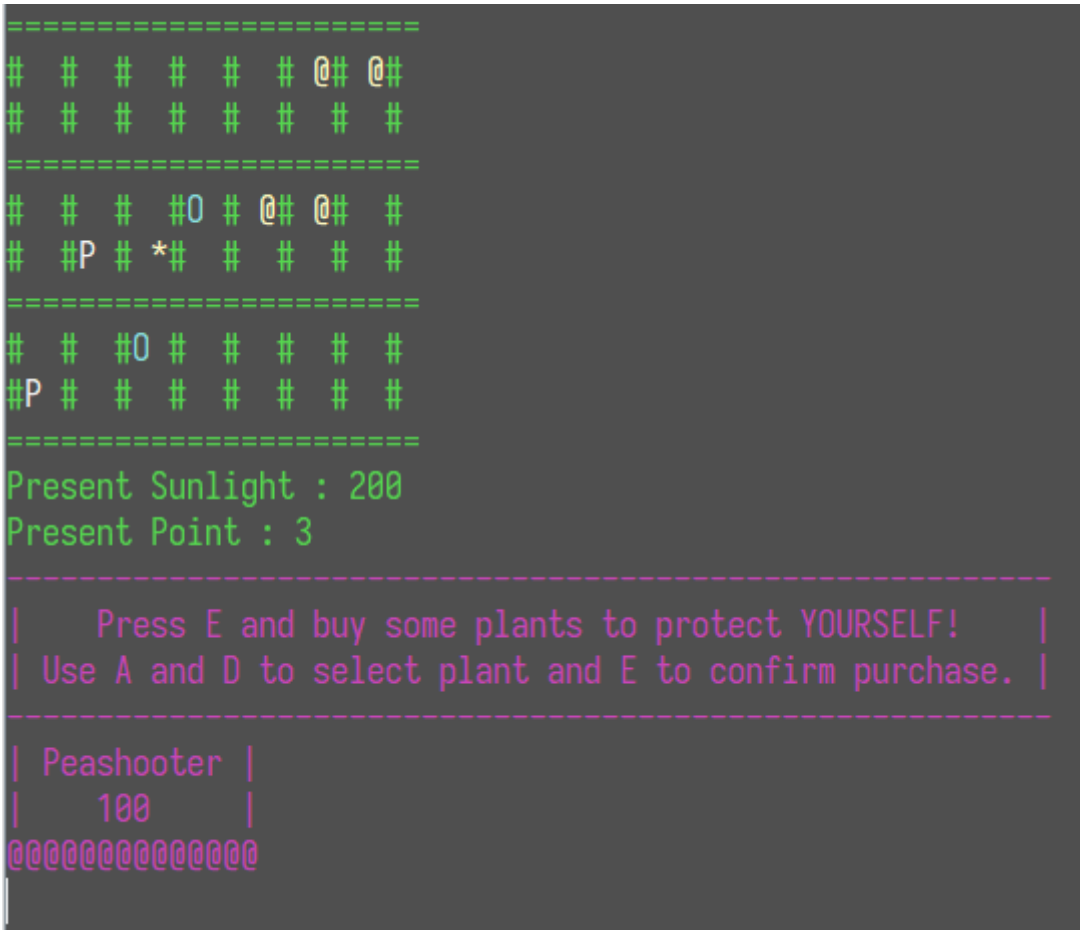
2. 游戏进行界面



按下E键进入游戏后

进入游戏后，系统开始自动生成阳光，同时地图上开始生成僵尸。

下面讲解游戏界面各部分的功能：



进行中的游戏

2.1 数值界面

数值界面显示了当前阳光和当前得分。

2.2 商店界面

按E键开始选择植物，A键D键控制左右移动，未被选中的植物下方默认为一行‘-’被选中的植物下方为一行‘@’，再次按E键确认选择植物，如果阳光数量足够进入地图选择环节。

2.3 地图界面

地图中每一个地块中各元素的相对位置为：

子弹格	僵尸格
植物格	光标格

- 子弹格：目前仅支持显示豌豆射手发射的青色子弹，用‘O’表示；
- 僵尸格：目前仅支持显示黄色的普通僵尸，用‘@’表示；
- 植物格：目前仅支持显示白色的豌豆射手，用‘P’表示；
- 光标格：表示当前地块已被选中，按下E键进行确认种植操作，如果该地块没有植物，则将选中植物种在该地块，用‘*’表示。

四、相关问题及解决方案

1. **问题**：起初的思路是建立一个主类Object，植物Plant、子弹Bullet和僵尸Zombie都是它的子类，在每个地块中只用三个基类指针，在遍历地块时只需循环调用三次有关方法即可，但是对此思路实现不是很理想。

   解决方案：重新构想类与类之间的关系，将操作提高到游戏Game层，底层类只存放有关数据以及数据的Get-Set方法，降低了底层类之间的耦合度，整体架构更加清晰。

2. **问题**：最开始时只想过用一个字符表示地块信息，但是后来发现对于植物、子弹和僵尸的相互作用判断在这种设计下变得十分麻烦。

   解决方案：将地块增加到四个字符的大小，各元素互不冲突，表达方式更加直观。

3. **问题**：在读取玩家操作符时无法同时进行屏幕打印或者无法在屏幕打印时读取玩家操作符。

   解决方案：采用多线程的思路，将打印操作放置在单独的线程中运行，在主线程中读取玩家的操作符。

4. **问题**：程序退出时报错abort() has been called，可能是销毁线程时遇到了问题。

   解决方案：由于使用的是C++11的thread类，类中没有封装exit方法，所以只能提前设定thread对象的join/detach属性，添加detach语句后程序不再报错。

5. **问题**：屏幕闪烁问题。

   解决方案：未解决，准备在图形化过程中使用双缓冲技术解决屏幕闪烁问题。