植物大战僵尸课程设计报告

一、课程设计的主要内容、目标和设计思路

主要内容：

实现基于命令行的简化版的植物大战僵尸小游戏，模仿设计前院场景，白天无尽模式，会不断产生僵尸，没有胜利条件，当任何一只僵尸进入左边庭院底线时，游戏结束，返回游戏玩家本次游戏的最终分数。

目标：

庭院场景实现3行7列，每个格子会显示当前格子里的植物的名字和血量、僵尸的名字和血量、僵尸数量，豌豆子弹等等。

系统每隔一段时间会自动产生自然光。

实现了两种僵尸：普通僵尸和路障僵尸，实现僵尸的生命值、攻击力、移动速度和攻击速度的属性。

实现了两种植物：豌豆射手和向日葵，实现植物的生命值、购买所花费的阳光数的属性和每种植物的特定功能。豌豆射手会每个时间周期发射一枚豌豆，向日葵会每个时间周期产生一定的阳光。

每块地块中可以有多只僵尸

实现植物的购买、地块的选择和植物的选择，设计光标以实现对用户友好。

实现当局游戏的记分牌，游戏分数会随着游戏时间的增加而增加，打死对应的僵尸也会获得不同的分数。

实现豌豆子弹的移动动画，颜色。

实现商店购买后的冷却功能。

设计思路：

采取面向对象的程序设计方法，将游戏所需要的各类元素封装成各种类，通过对象之间传递消息才实现对应的功能。基于游戏设计的主循环模式，先是绘制游戏画面，再获得玩家输入，然后更新游戏状态，再重新绘制游戏画面，循环持续到游戏结束。

二、主要类的设计

本次设计一共由6个类实现，分别是**Game类、Grid类、Store类、Plants类、Zombies类、Bullets类**，下面分别介绍：

Game类

```
class Game
{
public:
    Game();
    ~Game();
    //游戏的主循环
    void MainLoop();
    //绘制游戏的草坪的基本框架
    void DrawStage();
    //绘制游戏中的所有子弹
```

```
    void DrawBullets();
    //绘制游戏中的植物和僵尸信息等等
    void Draw();
    //实时活动用户输入并进行相应按键功能的实现
    void GetKey();
    //产生自然太阳光和向日葵产生太阳光
    void AddSunlight();
    //随着时间增加分数
    void AddScore();
    //移动网格光标
    void MoveGridCursor(int dx, int dy);
    //产生僵尸
    void AddZombie();
    //僵尸移动
    void MoveZombie();
    //游戏结束
    bool EndGame();
    //刷新每行是否存在僵尸的状态
    void RefreshExist();
    //所有的豌豆射手攻击
    void PeaShoot();
    //移动子弹
    void MoveBullets();
    //子弹攻击僵尸
    void AtkZombie();
    //僵尸攻击植物
    void AtkPlant();
private:
    State state;//标记游戏处于什么状态
    Store store;//生成一个商店对象
    Grid* grid;//一个grid类的指针，用于储存动态grid对象
    int cursor;//地块选择指针位于何处
    int sunlight;//游戏的当前阳光数
    int score;//游戏的当前分数
    Zombies _zombies[2];//游戏所有能产生的僵尸的数据模板
    vector<Bullet*> bullets[STAGE_LINES];//游戏每行草地子弹的容器
    bool exist_zombie[STAGE_LINES];//此行草地是否含有僵尸，用于确定是否发射子弹的条件之一
};
```

Game类中承当了游戏的各种功能的实现，统筹其他各种类，以实现游戏所需要的功能。

其数据成员包含：

1. 一个**枚举类型的State**状态变量，用于标记游戏当前处于什么状态，一共分为**BUYING、SELECT、GAMEOVER**三种状态，分别对应玩家正在商店选择购买植物、玩家正在选择植物种在哪个格子、游戏结束。
2. **Store类的对象**，用于向商店发送信息。
3. **Grid类对象的指针**，用于进行创建Grid类的动态对象，采用一维数组来代替二维数组的方式创建动态对象数组，以实现一个3*7大小的网格。
4. **cursor**用于储存当前游戏草坪的光标所在网格的下标。
5. **sunlight**表示当前游戏的阳光数。
6. **score**表示当前游戏的分数。
7. **_zombies[2]**数组储存了游戏中所有僵尸的数据模板，用于在生成僵尸的时候赋值给动态僵尸对象。
8. 一个**vector<Bullets>的向量容器**，用于储存每行中的豌豆子弹队列
9. **bool型的数组**，用于标记一行中是否含有僵尸，有为true，用于确定豌豆射手是否需要发射豌豆。

其成员函数包括：

Game中实现了**游戏的主循环**，采用一般游戏的主循环模式，绘制、输入、更新。

```
void Game::MainLoop()
{
    HideCursor();//隐藏光标
    SetCMDSize(CMD_COLS, CMD_LINES);//设置命令行尺寸
    while (true)//主循环
    {
        Draw();//绘制界面
        if (EndGame())//判断游戏是否结束
            break;

        GetKey();//实时获得输入
        store.AddBuyCD();//刷新商店cd

        AddSunlight();//增加阳光
        AddScore();//增加分数

        AddZombie();//产生僵尸
        MoveZombie();//移动僵尸
        AtkPlant();//攻击植物

        RefreshExist();//刷新每行是否存在僵尸的状态
        PeaShoot();//发射豌豆
        MoveBullets();//移动豌豆
        AtkZombie();//攻击僵尸
    }
}
```

Draw()函数：用于绘制游戏运行的画面，会调用DrawStage()，用于绘制游戏的基本网格边框、游戏操作提示、当前阳光数和游戏分数，然后调用store对象中的DrawStore()成员函数绘制商店界面，再遍历grid数组，调用Grid对象中的DrawGrid()成员函数绘制所有网格中的内容，最后调用DrawBullets()函数绘制豌豆子弹，基于层层覆盖的关系最终形成游戏画面，并且调用沙漏函数SandGlass实现0.2s刷新一次，保证游戏大概在5帧左右。

PS：

何为沙漏函数?

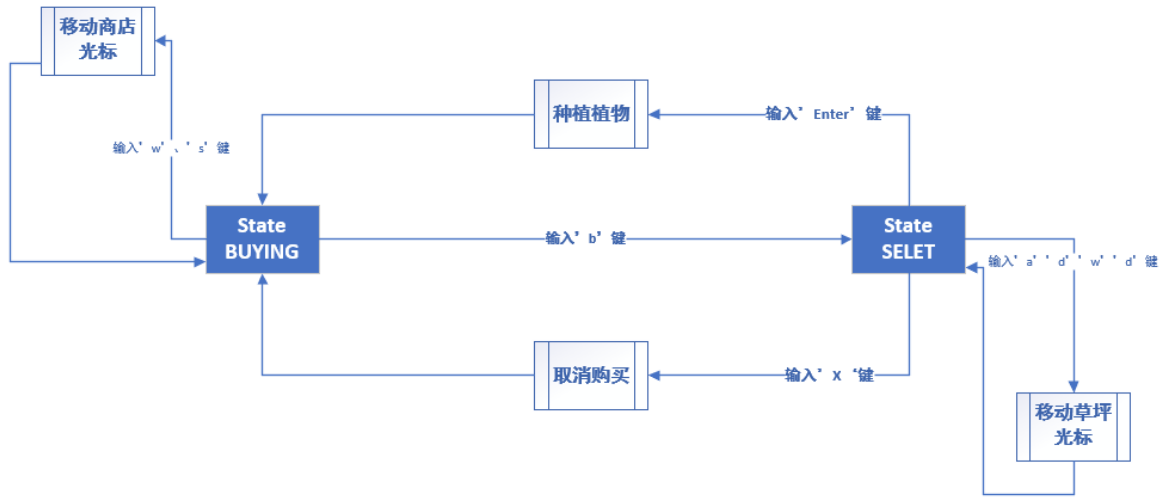
*这是一个在本程序中设计者自己编写的一个用于计时的功能函数，其位于UI.h头文件中，函数原型为bool SandGlass(clock_t& c, int s)，表示每s * 0.1秒放回一次true，用于实现周期性执行。*

为什么不实现更高帧率?

可以但是目前没必要，因为游戏中更新的最小周期为豌豆子弹的移动周期，也为0.2s，所有稳定在5帧左右已经能够基本显示豌豆的移动动画了，而且降低帧率可以减若闪屏感。

GetKey函数：通过库文件中的_kbhit()实时获得用户输入，不会阻断游戏执行，只有当用户输入时才会相应，并且设置了一个大小写转化过程，以便玩家在大写模式下的输入也能被执行。得到玩家的输入后，采用类似于状态机的模式做出反应，基本循环为：游戏刚打开state为BUYING，进入购买状态，此时通过'w'、's'键控制商店中光标的上下移动，当输入'b'键时，如果玩家阳光数足够并且植物未处于冷却状态时，state切换为SELECT，进入选择地块状态，此时关闭商店中的光标显示，光标显示切换到草坪中，否则不作为；当游戏处于SELET状态时，可以通过'a'、'd'、'w'、's'键控制光标移动，用于选择种植植物的格子，按'x'键可以取消种植，此时state又切换为BUYING，隐藏草坪的光标，重置并显示商店的光标，按'Enter'键确定种植，如果当前格子没有植物，则sunlight扣除植物的cost，state变为BUYING状态，启动植物的冷却时间，然后调用grid对象的GrowPlant函数。

流程图如下：



AddSunlight()函数：实现周期性增加阳光，包括自然光和向日葵产生的阳光。自然光产生周期为每6秒加25，向日葵为6秒加25。

AddScore()函数：实现每1秒增加1分。

MoveGridCursor(int dx, int dy)函数：实现草坪光标的移动，dx和dy表示偏移量，如果光标所在位置加上偏移量后越界，则将偏移量修正为0，然后光标加上修正后的偏移量，实现光标的移动。

AddZombie()函数：实现生成僵尸。采用clock()函数获得游戏已经执行的时间，根据游戏执行时间以不同的方式产生僵尸：前18秒不产生僵尸，18秒到78秒每隔10秒产生普通僵尸，78秒到98秒每隔9秒产生路障僵尸，98秒到103秒不产生僵尸，给玩家一个缓冲，103秒后到达最高难度，每2.5秒产生一个僵尸，僵尸种类随机。

MoveZombie()函数：遍历所有grid，如果僵尸可以移动（定义为移动周期达到并且没有植物阻挡），则让僵尸从该格子的容器中出来，并且进入下一个格子的容器中。如果位于x轴0处的僵尸可以移动，则将state变为GAMEOVER，表示游戏结束，接下来会在执行EndGame()时返回true，跳出主循环，游戏结束。

EndGame()函数：清空命令行，显示游戏结束信息。

RefreshExist()函数：及时更新每行是否含有僵尸的信息，以便豌豆射手做出反应。

PeaShoot()函数：遍历所有格子，如果豌豆射手可以发射子弹（定义为该行有僵尸并且植物内部的功能函数计时到0），则产生一个动态Bullets对象，放入对应的行的豌豆子弹容器中，每行的豌豆容器都以队列的方式储存豌豆。

MoveBullets()函数：遍历每一行，如果容器中的豌豆子弹可以移动（定义为移动计时为0），则调用Bullet的成员函数MoveBullet()移动子弹，如果返回true，说明子弹已经射出庭院，此时会将该豌豆子弹移出容器，归还豌豆子弹内存。

AtkZombie()函数：实现豌豆子弹打中僵尸。遍历每行，找出所有豌豆子弹坐标转换成网格坐标后对应网格有僵尸的豌豆子弹，调用格子容器中第一个僵尸的被攻击函数BeAttacked，如果返回true，说明僵尸被打死，则将此僵尸移出容器，归还内存，最后将已经打中的子弹移出子弹容器，归还内存。

AtkPlant()函数：实现僵尸吃植物。遍历所有grid，找出所有网格中既存在僵尸又存在植物的网格，先将僵尸的变为不可移动状态，然后如果僵尸可以攻击（定义为僵尸攻击计时为0），则调用对应Plants对象的Ate函数，如果返回true，表示植物死亡，再将这格僵尸容器里面的所有僵尸变为可移动状态。

PS：

为什么僵尸生成，僵尸移动，子弹生成，子弹移动等操作全在Game类中？

因为它们的容器都在grid里，而且只有通过game对象才能遍历grid，所有不得不在game中实现，可以考虑再增加一个类用于储存grid，从而更好的归还各类的功能，就目前来看Game类中达到功能有些过多了，由于时间有限，下次再修改吧。

Grid类

```
class Grid
{
public:
    Grid();
    void DrawGrid();//绘制格子中的内容
    void ChangCursor();//改变格子中的光标状态
    void GrowPlant(Plants& p);//在格子中种植植物
    bool Is_Grow();//得到该格子是否可以种植植物的信息
    void SetGridXY(int _x, int _y);//设置格子的坐标
    vector<Zombies*> GetVector();//返回存储僵尸的容器
    bool ExistZombieOrPlant();//返回当前格子是否有僵尸或者植物
    bool ExistZombieAndPlant();//返回当前格子是否有僵尸和者植物
    friend class Game;
private:
    Plants plant;//当前格子的植物
    vector<Zombies*> zombies;//一个储存僵尸对象指针的容器
    int x;//（x，y）代表格子的左上角在命令行中的坐标
    int y;
    int zombie_nums;//当前格子的僵尸书
    bool is_cursor;//是否光标在这
```

```
    bool is_grow;//是否可以种植，同时作为是否有植物存在的标志
};
```

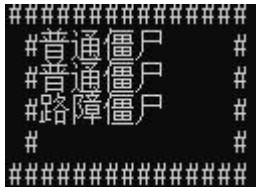
数据成员：

1. **plant**用于用处储存该格子的植物对象，因为一格只有一个植物，所有只需一个对象实体即可。
2. **vector<Zombies*>**类的容器用于储存该格子中的僵尸，因为一个格子有多个僵尸，所有僵尸采用动态对象，容器内储存其指针。
3. **（x， y）**表示网格左上角的坐标，用于绘制定位。

4. **zombie_nums**表示该格子的僵尸数，用于显示信息，判断僵尸有无，访问容器时不越界。
5. **is_cursor**表示该格子是否处于光标选择状态，用于判断是否需要绘制光标。
6. **is_grow**表示该格子是否可以种植植物，所有格子初始化为可以，同时可以表示是否含有植物，死亡时只需要将格子中的植物数据清理，is_grow改为true即可实现死亡植物不显示，再次种植则对plant赋值即可。

成员函数：

DrawGrid()函数：用于绘制格子信息，分为是否需要绘制光标，植物名字和HP，僵尸植物和HP，当一块格子有多个僵尸时，不显示僵尸HP，当僵尸数大于4时，会显示网格僵尸数，当植物于僵尸重合时，不显示僵尸HP，显示僵尸数目。



ChangCursor()函数：改变网格光标标记状态，将is_cursor的bool值取反。

GrowPlant(Plants& p)函数：种值植物，将商店传来的模板赋值给动态Plants对象。

Is_Grow()函数：返回当前格子的is_grow状态。

GetVector()函数：用于得到格子里的僵尸容器。

ExistZombieOrPlant()函数：用于判断格子是否含有僵尸或者植物，主要用于绘制子弹时，如果格子有植物或者僵尸则不需要显示，防止子弹覆盖植物。

ExistZombieAndPlant()函数：用于判断格子是否含有僵尸和植物，主要用于判断是否需要进行僵尸吃植物的处理。

这里声明**Game类为Grid的友元**，是应为Game类中很多地方需要使用Grid的数据成员，如果每次都通过接口实现代码过于冗长。

Store类

```
class Store
{
public:
    Store();
    //绘制商店
    void DrawStore();
    //移动光标
    void MoveCursor(int dy);
    //当进入选择种植地点时进入隐藏商店光标状态
    void HideCursor();
    //重置光标
    void ResetCurset();
    //让cd归零，开始计时
    void RefreshBuyCD();
    //增加cd的计时
    void AddBuyCD();
    //检测冷却时间是否结束
    bool CDIsOk();
    //得到光标所指向的植物的所应花费发阳光数
    int GetCost();
    //售卖植物
    Plants& SellPlant();
private:
    Plants plant_card[PLANT_KINDS];//植物的模板
    bool is_cursor[PLANT_KINDS];//光标是否在这
    int buy_cd[PLANT_KINDS];//每种植物的cd
    int cursor;//光标位置
};
```

数据成员：

1. 一个**Plants类型的数组**，用于储存植物的各种数据的模板，可以通过Plants类中的PlantSet函数进行设置。
2. **is_cursor**表示数组是否在这，用去确定是否需要绘制光标
3. **int类型buy_cd数组**用于储存植物的冷却进度。
4. **cursor**表示商店的光标在哪，他是植物模板数组的下标，会让光标显示在对于植物处。

成员函数：

DrawStore()函数：绘制商店，会显示植物名字，植物价格，植物冷却时间，光标，如图：

CDIsOk()函数：用于判断是否cursor所在的植物是否已冷却完，用于在购买植物时做判断能否购买。

MoveCursor(int dy)函数：移动光标，dy是光标的偏移量，如果cursor+dy越界，则dy置为零，再然后cursor加上重置后的dy。达到移动光标和防止越界的目的。

HideCursor()函数：隐藏光标，当进入SELET状态后，将不再显示光标。

ResetCurset()函数：重置光标，当用SELET进入BUYING时，将重置光标，重新开始显示光标。

AddBuyCD()函数：增加cd的冷却计时。

RefreshBuyCD()函数：将cd置为零。

SellPlant()函数：将光标指向的植物的模板数据输出，用于赋值。

Plants类

```
class Plants
{
public:
    Plants();
    //用于从商店设置各种植物的参数
    void PlantSet(const char* _name, int _hp, int _cost, int _buycd, int
_functioncd, bool _shoot, bool _addsunlight);
    //得到植物的名字
    const char* GetPlantName();
    //得到植物的购买冷却时间
    int GetPlantBuyCD();
    //得到植物购买所需的花费
    int GetPlantCost();
    //得到植物的血量
    int GetPlantHP();
    //实现对=的重载，分别植物的种植时从商店的模板赋值
    Plants& operator = (const Plants& p);
    //植物被僵尸吃，扣除hp，如果植物被吃掉放回true
    bool Ate(int atk);
    //如果植物具有发射豌豆功能，则调用发射豌豆
    Bullet* Shoot(int x, int y);
    //如果植物具有产生阳光功能，则调用产生阳光
    int AddSunlight();
private:
    const char* name;//植物名字
    int hp;//植物血量
    int cost;//植物所需的阳光
    int buycd;//植物的购买冷却时间
    int functioncd;//植物的功能函数执行的冷却时间
    clock_t c;//植物的计时器
    bool shoot;//植物是否具有发射功能
    bool addsunlight;//植物是否具有产生阳光的功能
};
```

植物的数值：

name	HP	COST	BUY_CD	FUNCTION_CD
豌豆射手	200	100	5秒	1.5秒/颗
向日葵	200	50	5秒	6秒/25阳光

数据成员：
如注释所著，大部分比较好理解。提一下：

1. **clock_t类型的c**用于配合沙漏函数进行计时，如果计时器满了，则会植物会执行对应功能。
2. **shoot和addsunlight**分别是用于标记植物是否可以发射豌豆子弹和产生阳光。因为豌豆射手和向日葵都是一个类，所以用此区分植物对象分别具有什么功能。

为什么不设置植物的攻击力?
因为目前的植物不需要，将来也未必需要，向日葵没有攻击力，而豌豆射手的攻击则是以子弹Bullets来体现的，即使未来增加窝瓜，樱桃炸弹等等，其杀死僵尸功能完全可以通过功能函数来实现，也没必要设置攻击力这一属性，故这里不设置植物攻击力的体现。

成员函数：

PlantSet(const char* _name, int _hp, int _cost, int _buycd, int _functioncd, bool _shoot, bool _addsunlight)函数：用于设置植物模板的各种参数。

一些Get函数：得到对应需要的量。

Plants& Plants::operator = (const Plants& p)函数：重载操作符=，用于在种植植物时利用商店植物的模板为grid上的plant赋值。

Ate(int atk)函数：植物收到攻击时会调用的函数，植物的HP会减去攻击力，如果植物的HP减为0，则会返回true，用于判断是否进行接下来的消亡处理。

Shoot(int x, int y)函数：植物的功能函数——发射豌豆。如果达到发射子弹的条件，会根据传入的坐标在对应坐标创建一个Bullets动态对象。

AddSunlight()函数：植物的功能函数——生产阳光。如果达到时间，则会产生25点阳光。

Zombies类

```
class Zombies
{
public:
    Zombies();
    Zombies(const Zombies& z);
    //用于在game类中设置所有僵尸的数值
    void SetZombie(string _name, int _hp, int _atk, int _speed, int _score);
    //得到僵尸的名字
    string GetZombieName();
    //得到僵尸的血量
    int GetZombieHP();
    //僵尸本次循环僵尸是否能够移动
    bool ZombieMove();
    //僵尸本次循环僵尸是否能够攻击植物
    bool ZombieEat();
    //在产生僵尸时初始化僵尸的计时器
    clock_t& GetMoveCounter();
    clock_t& GetEatCounter();
    //得到僵尸的速度
```

```
    int GetZombieSpeed();
    //得到僵尸的攻击力
    int GetZombeAtk();
    //得到僵尸被打死所能得到的分数
    int GetZombieScore();
    //修改植物是否可以移动的状态
    bool& GetMove();
    //僵尸被攻击的处理函数
    bool BeAttacked(int atk);
private:
    string name;//僵尸名字
    int hp;//僵尸血量
    int atk;//僵尸攻击力
    int speed;//僵尸速度
    int score;//僵尸分数
    bool move;//僵尸是否可以移动
    clock_t move_counter;//僵尸的移动计时
    clock_t eat_counter;//僵尸的攻击计时
};
```

僵尸数据：

NAME	HP	ATK	SPEED	SCORE
普通僵尸	200	25	8秒	25
路障僵尸	570	25	8秒	40

数据成员：

1. **采用string类储存名字**，因为僵尸是通过动态对象保存的，会有消亡问题，如果使用const char*则还需要重写拷贝构造函数，不然就会出现对同一块内存进行多次释放的问题，所以直接用string类来表示。
2. **move**用于标记僵尸能否移动，当僵尸与植物重合时，僵尸将在吃掉植物前都不能移动，无论其移动计时是否达到。
3. **move_counter和eat_counter**分别是僵尸进行移动和吃植物的计时器。

成员函数：

SetZombie(string _name, int _hp, int _atk, int _speed, int _score)函数：用于在Game类中设置僵尸数值模板。

ZombieMove()函数：判断僵尸能否移动，一是满足计时达到，二是move处于true，即没有植物阻挡。

ZombieEat()函数：判断僵尸是否可以攻击，即计时达到，这里设置的是2秒一次攻击。

BeAttacked(int atk)函数：僵尸被攻击时执行该函数，hp减去对应atk，如果hp归零，则返回true，接下来对已死亡僵尸进行消亡处理。

Bullets类

```
class Bullet
{
public:
    Bullet(int x, int y);
```

```
    void DrawBullet();//绘制子弹
    bool MoveBullet();//移动子弹
    int GetBulletX();//得到子弹所在x坐标
    int GetBulletY();//得到子弹所在y坐标
    int GetBulletAtk();//得到子弹的攻击力
private:
    int x;//子弹坐标初始位置
    int y;//子弹所在y轴
    int dx;//子弹x轴偏移量
    int atk;//子弹攻击力，单位：点/每颗
    int speed;//子弹移动速度，单位：100ms/每格
    clock_t c;//用于移动计时
};
```

数据成员：

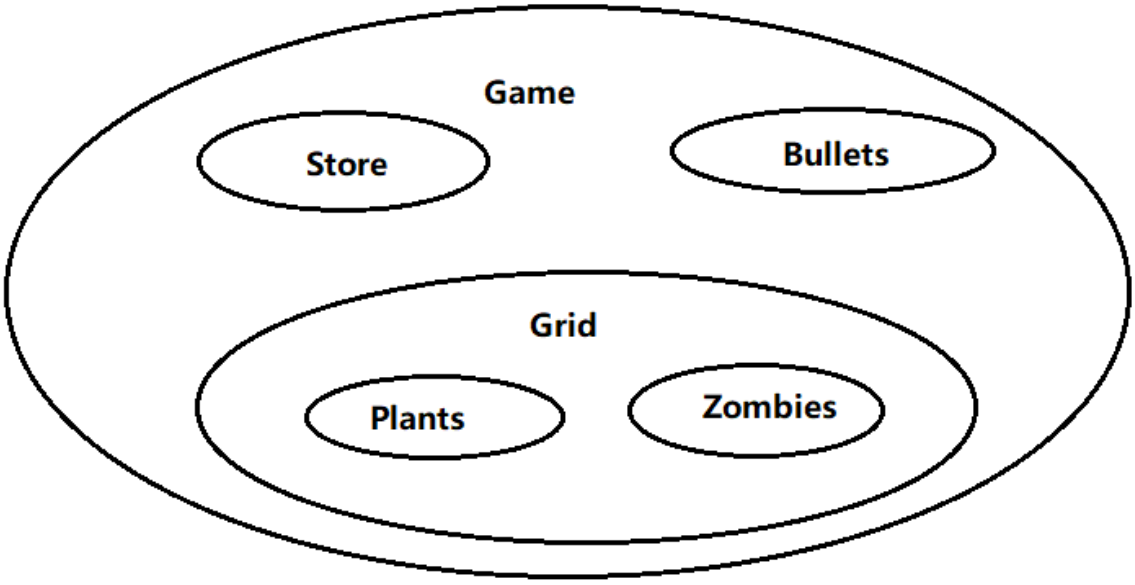
1. **（x，y）**表示子弹在命令行中的初始坐标，而**（x + dx，y）**才是子弹在命令行中的当前坐标。
2. **speed**表示子弹速度，这里设置为0.2秒移动一次，一次dx增加2
3. c用于子弹移动的计时器。

成员函数：

DrawBullet()函数：绘制子弹图标，并且通过更改命令行输出字体颜色，将子弹改为绿色。

MoveBullet()函数：实现子弹的移动，如果子弹射出庭院，返回true，对该子弹进行消亡处理。

类之间的关系



Game类作为游戏的主体，负责协调其他类，向其他类发送信息实现对应功能。

Grid类中包含Plants类和Zombies，要对Plants和Zombies操作必须通过Grid。

Store类和Bullets类不基于Grid进行操作，但是需要Game对他们发送信息来实现对应功能。

三、程序的亮点与运行操作方法

亮点

1.利用命令行的输出文本改变字体颜色设置光标，达到对用户友好化界面。

2.实现了豌豆子弹的动态移动，形成了一定的动画。

3.游戏难度设置合理，不会太简单，也不会太难。

运行操作方法

双击P_V_Z.exe文件打开程序后，可以用“w”“s”键控制光标移动，选择好要购买的植物后，输入“b”进行购买，进行地块的选择，此时可用“w”“s”“a”“d”控制光标上下左右移动，输入“x”表示取消购买，输入“Enter”表示确定种植。请构筑好自己的防线，游戏难度会越来越大，保护好你的脑子！

四、遇到的问题与解决方法

1.游戏会有闪屏现象，部分文本一闪一闪的。

经过察觉，造成问题的原因在于大量面积的清屏操作，大面积的清屏就会有一个闪烁的现象。优化方案有两个，一个是设置Draw的调用频率，也就是降低画面刷性速度，这里设置的是1秒绘制5次；二是修改绘制方法，由原来的清屏操作改为移动命令行的输出指针，隐藏指针后采用输出覆盖，将新的信息覆盖在原画面上，可以有效降低画面闪烁感。

2.豌豆的子弹输出时会相互覆盖

因为用户种植豌豆射手和投放僵尸是不确定的，当一行有多个豌豆射手时，豌豆子弹占2个字节，所以输出会相互覆盖，导致一些子弹只剩下半颗。目前还没有较好的解决方法，因为这个问题有一定的随机性，也许当学会使用图片素材时可以解决这个问题，由于图片是透明的，在绘制的时候就会出现重叠，而不是像现在这样由于非透明而覆盖。