Report

姓名：唐业

学号：**181860087**

院系：计算机科学与技术系

一、概述

1．主要内容

实现命令行下简易的植物大战僵尸小游戏——白天无尽模式下的前院场景。每种僵尸将按照特定的时间间隔，随机出现在马路的任一行，以庭院左边的底线为目标前进。玩家通过收集阳光（向日葵产生阳光将自动收集），种植植物以及攻击僵尸来保护房子。一旦任何一只僵尸进入庭院左边底线，游戏结束玩家失败。

2．目标与设计思路

1）界面设计

游戏界面，地图共 5 行 10 列，包括马路的一列。下方有商店、阳光计数和计分板。

由于是字符界面，每个地块只显示下面所示的其中一类：

一株植物，可附加其额外状态信息

一只僵尸，可附加其额外状态信息

多只僵尸：可以考虑每个僵尸写一行，这样可以容纳四个僵尸，更多的情况直接忽略类别写作僵尸 x5

不考虑植物和僵尸共处一格的情况，假设植物会阻挡僵尸进入那一格，直到僵尸把它啃完才可以。因此某些植物将无法实现（地刺、土豆地雷）。

设计的植物种类包括：豌豆射手、双发射手、西瓜投手、坚果墙、高坚果、窝瓜、樱桃炸弹、火爆辣椒、向日葵、大蒜

设计的僵尸类型包括：普通僵尸、路障僵尸、铁桶僵尸、摇旗僵尸、橄榄僵尸、小丑僵尸、读报僵尸、撑杆僵尸、投石僵尸

对于子弹，豌豆以及投石僵尸的篮球用“●”表示，西瓜用★表示，将沿一定路径横向移动，行进动画通过位置变化界面刷新实现。

2）交互设计

捕获键盘字母 b 进入商店，通过上下左右键选择要买的植物，按回车确认，按 ESC 取消，若确认，焦点将移动到庭院地图，通过上下左右选择地块。确认后，如果对应位置能够种植，则扣除相应阳光值。

捕获键盘字母 c 进入铲子模式，通过上下左右键选择要铲除的地块。确认后，若对应位置有植物，则执行铲除。

3）整体框架

Controller 成员函数 StartGame()中利用头文件 conin.h 中的_kbhit()和_getch()两函数非阻塞的获取键盘字符输入。每轮循环休眠 CYCLETIME=100ms 来进行循环读取，并在每次循环中重绘四个部分：庭院地图中需要重绘的格子、商店列表、计分板和阳光计数，以及调用模型类的 Main()函数，其中既对每个已有对象调用各自的 Main()，根据各对象 Main()返回值进行相应操作。又同时执行商店列表的冷却过程、僵尸出场的“冷却过程”。

二、 具体设计

1．设计模式

采用MVC模式，模型（Model）包括植物类、僵尸类和子弹类，视图（View）将根据模型数据，刷新绘制界面，控制器（Controller）将捕获用户输入，调控模型（Model）和视图（View）完成植物种植或铲除。

2．主要类

工具类：

1）用于从文件中读入各类型对象信息后，存入ObjInfo类内保存

```
typedef struct stObjectInfo {
    OBJECT_TYPE type;
    int hp;
    int speed;
    int attack;
    int coolingTime;
    int cost;
    int score;
    stObjectInfo(OBJECT_TYPE _type, int _hp, int _speed, int _attack, int
_coolingTime, int _cost, int _score);
}ObjInfo, *PObjInfo;
```

基本每一项对应从.csv文件（后文详细解析结构）读入的数据，存入该类型中方便读取用于构造继承自Base类的对象。

2）用于存储庭院地图每个格子的状态，存入ObjMapItem类的二维数组内

```
typedef struct stObjMap {
    OBJECT_TYPE objType;
    LPCWSTR name;
    int coolingTime;
    int cost;
    int cnt;
    bool negative;
    bool repaint;
    stObjMap();
    stObjMap(OBJECT_TYPE _objType, bool _negative) :objType(_objType),
cnt(0), negative(_negative);
    stObjMap(OBJECT_TYPE _objType, LPCWSTR _name, int _coolingTime, int
_cost, bool _negative, bool _repaint);
}ObjMapItem, *PObjMapItem;
```

用于表示地图某一个块，或商店列表某一项，negative成员表示是否应该反色，repaint成员表示是否应该重绘。

3）自定义的Point类，重载必要的运算符便于后续运算表示坐标，兼容COORD类型

```
class Point
{
public:
    Point():X(0), Y(0){}
    Point(int x, int y) :X(x), Y(y) {}
    Point(const Point &p) :X(p.X), Y(p.Y) {}
    Point(const COORD &cr) :X(cr.X), Y(cr.Y) {}
    friend const Point operator+(const Point &p1, const Point &p2);
    friend const Point operator-(const Point &p1, const Point &p2);
    friend const Point operator*(const Point &p1, const Point &p2);
    friend bool operator==(const Point &p1, const Point &p2);
    Point &operator=(const Point &p);
    Point &operator+=(const Point &p);
    Point &operator-=(const Point &p);
    Point &operator*=(const Point &p);
    int X;
    int Y;
};
```

该类是对COORD结构体的一个拓展，重载了基本的算数运算符和逻辑运算符，便于后面表示坐标，进行运算等。

植物类、僵尸类、子弹类均继承自一个基类：

```
class Base
{
public:
    Base(LPCWSTR name, const Point &point, OBJECT_TYPE otype, OBJECT_TYPE
btype, int frames, int cnt);
    virtual ~Base() {}
    virtual bool isDead(void) { return m_hp <= 0; }
    virtual bool Move(MOVE_TYPE type) = 0;
    virtual int BeAttacked(const Base *obj) = 0;
    virtual Base *Attack(void) = 0;
    virtual int Main(void) = 0;
public:
    const Point &position(void) const { return m_point; }
    OBJECT_TYPE objectType(void) const { return m_objectType; }
    OBJECT_TYPE bulletType(void) const { return m_bulletType; }
    int hp(void) const { return m_hp; }
    int speed(void) const { return m_speed; }
    int attack(void) const { return m_attack; }
    int coolingTime(void) const { return m_coolingTime; }
    int cost(void) const { return m_cost; }
    int scroe(void) const { return m_score; }
    int frames(void) const { return m_frames; }
    int cnt(void) const { return m_cnt; }
    bool isNegative(void) const { return m_negative; }

    void setHp(int value) { m_hp = value; }
    void setSpeed(int value) { m_speed = value; }
    void setAttack(int value) { m_attack = value; }
    void setCoolingTime(int value) { m_coolingTime = value; }
    void setCost(int value) { m_cost = value; }
    void setScroe(int value) { m_score = value; }
    void setFrames(int value) { m_frames = value; }
    void setCnt(int value) { m_cnt = value; }
    void setNegative(int value) { m_negative = value; }
protected:
    LPCWSTR m_name;
    Point m_point;
    OBJECT_TYPE m_objectType;
    OBJECT_TYPE m_bulletType;
    int m_hp;
    int m_speed;
    int m_attack;
```

```
    int m_coolingTime;
    int m_cost;
    int m_score;
    int m_frames;
    int m_cnt;
    bool m_negative;
public:
    static const PObjInfo GetInfo(OBJECT_TYPE type);
};
```

其中静态函数 GetInfo 用于读取获得指定类型对象的信息对象，用于 Base 构造函数中的初始化。私有成员 m_frames 和 m_cnt 组合，通过 m_cnt 的计数以及和 m_frames 的对比决定对象（植物、僵尸和子弹）是否执行动作。私有成员 m_coolingTime 和 m_cnt 组合，实现植物商店列表冷却计数。四个纯虚函数，将强制要求在植物类、僵尸类和子弹类这些继承自 Base 类的对象中重写（部分接口继承）：

```
    virtual bool Move(MOVE_TYPE type) = 0;
    virtual int BeAttacked(const Base *obj) = 0;
    virtual Base *Attack(void) = 0;
    virtual int Main(void) = 0;
```

其中 Attack 函数返回一个向上造型为 Base*类型的子弹类指针,如果发出攻击的对象是用子弹类型攻击的（如豌豆、投石僵尸的篮球等）。

视图类：

```
class View
{
public:
    View();
    virtual ~View();
    void DrawFrame();//绘制地图框架
    void DrawScore(int score);//绘制计分板
    void DrawSunCnt(int cnt);//绘制阳光计数
    void DrawStore(const std::vector<PObjMapItem> &storeMap);//绘制商店列表
    void DrawMap(const Model &model);//绘制庭院地图格子
    void DrawStoreSelect(const std::vector<PObjMapItem> &storeMap, const Point &p);//绘制商店某位置为选中状态
    void ClearStoreSelect(const std::vector<PObjMapItem> &storeMap);//清空选中状态
    void DrawMapSelect(const Point &p);//绘制地图某位置选中状态
    void ClearMapSelect();//清空地图绘制状态
    void InvalidateBlock(Point leftTop, Point rightBottom);//重绘某方形区域，给出左上右下坐标
```

```
    void InvalidateMap(const Point &p);//重绘地图某位置
    void InvalidateBulletTrace(const Point &p);//重绘地图某位置中的子弹轨迹
public:
    void SetCursorPos(short x, short y);//设置光标位置
    void SetCursorPos(const Point &p);//设置光标位置
    void SetCursorPosRe(short reX, short reY);//设置光标相对位置
    Point GetCursorPos();//获取光标位置
private:
    HANDLE m_hOutput;
    Point m_cursorPos;
    Point m_mapPos;
    Point m_storePos;
    Point m_scorePos;
    Point m_sunCntPos;
    Point m_preStorePos;
    Point m_preMapPos;
    int getColor();//获取绘制颜色
    void setColor(int c);//设置绘制颜色
    void gotoxy(int x, int y);//设置光标位置
    void drawtext(int x, int y, LPCWSTR s);//在某位置绘制字符串
    void modeset(int w, int h);//调整窗口尺寸以及窗口缓冲区大小
    void InitPainter();//初始化绘制状态
    void drawObj(const Base *model, const Point &p);//绘制地图某位置所有对象
    void drawZombies(const std::vector<Base*> &zombies, const Point &p);//绘制某位置所有僵尸
    void drawStoreSelect(const std::vector<PObjMapItem> &storeMap, const Point &p);//绘制商店选中时反色状态
    void drawStoreUnselect(const std::vector<PObjMapItem> &storeMap);//绘制商店未选中
    void drawMapSelect(const Point &p);//绘制地图选中状态
    void drawMapUnselect();//绘制地图未选中状态
};
```

视图类中封装了对控制台界面操作的底层函数，包括光标位置、颜色设置、窗口大小和缓冲区大小、在某位置输出字符串等。同时提供了根据模型类以及模型类中组合的对象绘制庭院地图、商店列表、计分板、阳光计数的方法，其中各部分的起始绘制坐标（一般为左上角）在第一次运行绘制庭院地图、下方信息列表框架时确定。

模型类：

```
class Model
{
public:
    Model();
    virtual ~Model();
    bool NewObject(OBJECT_TYPE objType, const Point &point);//创建一个给定类型的对象
    bool DelPlants(const Point &p);//铲除植物
    void ClearCoolingState();//作弊接口：清除当前冷却状态
    int Main(void);
public:
    int sunCount() const;
    int scoreCount() const;
    const std::vector<Base*> &PlantsObj() const;//存储所有植物对象
    const std::vector<Base*> &ZombieObj() const;//存储所有僵尸对象
    const std::vector<Base*> &BulletObj() const;//存储所有子弹对象
    const std::vector<PObjMapItem> &StoreMap() const;//存储商店列表对象
    const std::vector<std::vector<PObjMapItem> > &ObjMap() const;//存储地图每个块的信息的二维数组对象
    void incSunCount(int value);//增加阳光
    void SetStoreMapItemNegative(unsigned int index);//设置商店项选中状态
    void ResetStoreMapItemNegative();//重置商店项为未选中状态
    void SetObjectMapItemNegative(const Point &p);//设置地图块为选中状态
    void ResetObjectMapItemNegative();//重置地图块为未选中状态
    bool isZombie(const Point &p) const;//判断某位置是否有僵尸
    bool isPlant(const Point &p) const;//判断某位置是否有植物
    bool isBullet(const Point &p) const;//判断某位置是否有子弹
    Base *getZombie(const Point &p) const;//获取某位置的首个僵尸指针（可能不止一个）
    Base *getPlant(const Point &p) const;//获取某位置的植物指针
    Base *getBullet(const Point &p) const;//获取某位置的首个子弹指针（可能不止一个）
private:
    int m_sunCnt;
    int m_score;
    std::vector<Base*> m_vPlantsObj;
    std::vector<Base*> m_vZombieObj;
    std::vector<Base*> m_vBulletObj;
    std::vector<std::vector<PObjMapItem> > m_vObjMap;
    std::vector<PObjMapItem> m_vStoreMap;
```

```
    std::vector<PObjMapItem> m_vZombieMap;
    void InitObjMap();//初始化地图对象二维数组
    void GenerateStoreMap();//生成商店列表
    void GenerateZombieMap();//生成僵尸列表
    bool isCooling(OBJECT_TYPE objType) const;
    bool isSunEnough(OBJECT_TYPE objType) const;
    bool isMapOccupied(const Point &p) const;
    void delPlant(const Base *obj);
    void delZombie(const Base *obj);
    void delBullet(const Base *obj);
    static bool Readin(void);
};
```

模型类中用 vector 存储植物类、僵尸类和子弹类，并提供了相应的管理、创建、删除接口，由于数量不会太多，因而采用 vector 数组存储，需要使用时遍历 vector 数组，根据对象在地图中的位置取出执行响应操作。

模型类的 Main()函数中，对每种对象的列表进行遍历，并调用每个对象的 Main()函数，根据返回值进行相应操作。同时完成商店列表的冷却过程，和僵尸随机生成的“冷却过程”。

控制器类：

```
class Controller
{
public:
    Controller();
    virtual ~Controller();
    int StartGame();
private:
    View m_cui;
    Model m_data;
};
```

控制器类中，StartGame()执行非阻塞循环轮调读取可能的用户键盘输入，每次循环中重绘四个部分：庭院地图中需要重绘的格子、商店列表、计分板和阳光计数，以及调用模型类的 Main()函数。同时失败条件将在 Main()返回非零时成立。

```
m_cui.DrawMap(m_data);
m_cui.DrawStore(m_data.StoreMap());
m_cui.DrawScore(m_data.scoreCount());
m_cui.DrawSunCnt(m_data.sunCount());
if (m_data.Main())
{
    printf("Game Over.\n");
    system("pause");
    exit(0);
}
```

三、程序功能

程序第一步将读取 data 文件夹下名为 modelInfo.csv 的数据文件。数据每一列含义为（其中某些类型未实现）：

类型标识，生命值，速度，攻击，冷却时间，消耗，分数

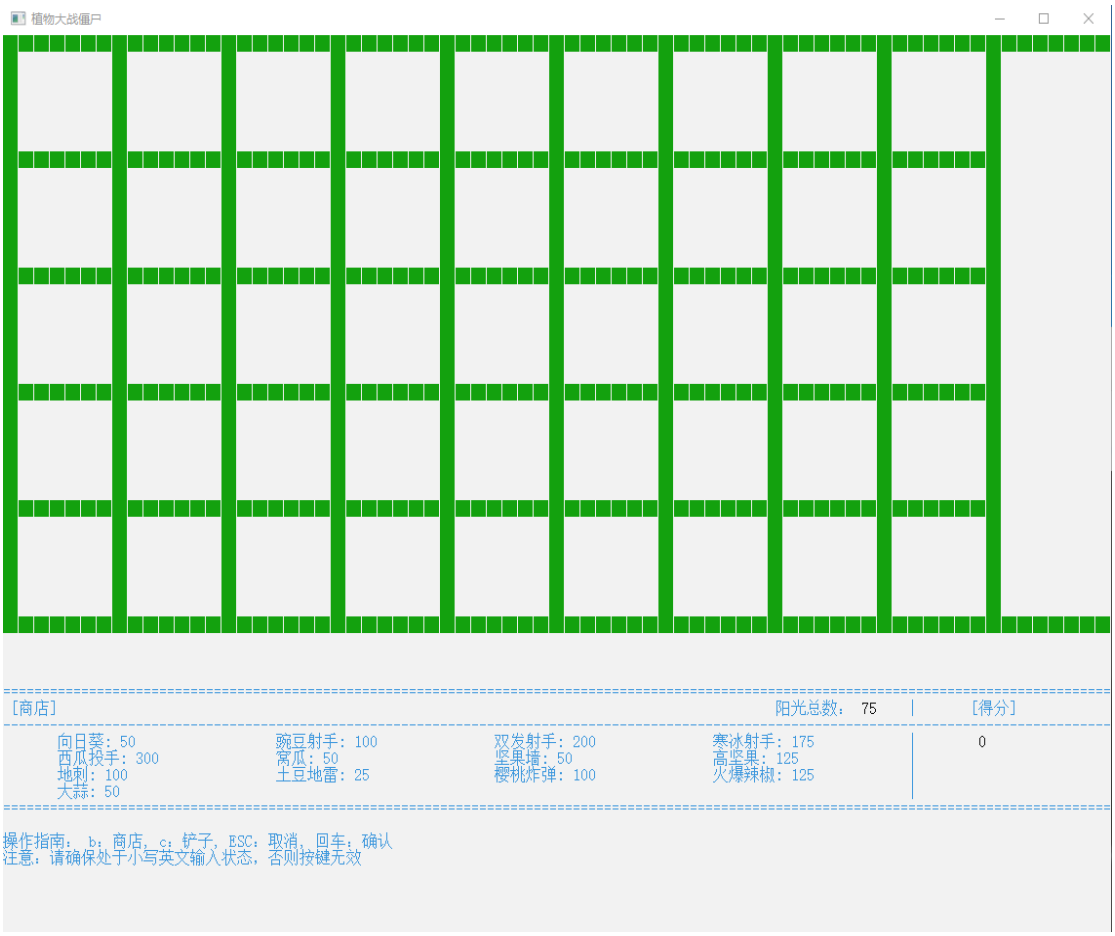
ZOMBIE_NORMAL	10000	1	600	25	0	10
ZOMBIE_CONEHEAD	28000	1	600	40	0	28
ZOMBIE_BUCKETHEAD	66000	1	600	50	0	66
ZOMBIE_SCREENDOOR	66000	1	600	60	0	66
ZOMBIE_FLAG	10000	1	600	45	0	10
ZOMBIE_FOOTBALL	81000	1	600	80	0	81
ZOMBIE_JACKINTHEBOX	17000	1	600	60	0	17
ZOMBIE_NEWSPAPER	18000	1	600	45	0	18
ZOMBIE_POLEVAULTING	17000	1	600	60	0	17
ZOMBIE_CATAPULT	35000	1	99999	90	0	35
ZOMBIE_DANCING	0	1	4000	0	0	9
ZOMBIE_BACKUP	0	1	4000	0	0	9
PLANT_PEASHOOTER	15000	1	0	8	100	0
PLANT_REPEATER	15000	1	0	8	200	0
PLANT_SNOWPEA	15000	1	0	8	175	0
PLANT_MELONPULT	15000	1	0	8	300	0
PLANT_SPIKEROCK	15000	1	0	8	100	0
PLANT_WALLNUT	210000	1	0	30	50	0
PLANT_TALLNUT	420000	1	0	30	125	0
PLANT_SQUASH	15000	1	0	10	50	0
PLANT_POTATOMINE	15000	1	0	30	25	0
PLANT_CHERRYBOMB	15000	1	0	35	100	0
PLANT_JALAPENO	15000	1	0	50	125	0
PLANT_SUNNER	15000	24	25	10	50	0
PLANT_GARLIC	60000	1	0	8	50	0
BULLET_ZOMBIE	1	6	0	0	0	0
BULLET_BASKETBALL	1	6	0	0	0	0
BULLET_SUN	1	6	50	0	0	0
BULLET_NORMAL	1	6	1000	0	0	0
BULLET_TWO	1	6	1000	0	0	0
BULLET_SNOW	1	6	1000	0	0	0
BULLET_MELON	1	6	1000	0	0	0
BULLET_SECKILL	1	6	99999	0	0	0

注意：

植物攻击力由发出的子弹攻击力表示；

僵尸的冷却时间列 x 为周期，表示该类僵尸每隔 x 秒随机产生一个；

界面如下：



捕获键盘字母 b 进入商店，捕获键盘字母 c 进入铲子模式，通过上下左右键进行选择，按回车确认。僵尸将会从最右方随机一个位置出现，每种僵尸均按照一定周期出现。

阳光设定每 5 秒自然产生 25 阳光，每杀死一个僵尸，将得分增加相应的值。

四、问题与解决方案

Q1：植物商店区的绘制，实时反馈冷却状态（如果有冷却中的）以及购买时选中状态？

A1：对于商店区的绘制，每次轮调循环都全部重绘一次，冷却状态由商店列表对象的两成员——冷却时间 m_coolingTime、计数 m_cnt 决定，选中状态由商店列表对象的成员 m_negative 确定，在视图类 View 重绘时检测上述指标判断绘制情况。

Q2：商店区、阳光计数、计分板位置的确定？

A2：只能在绘制上面庭院地图边框，下面的边界线时动态确定，当然有试的过程。

Q3：庭院地图块的状态确定：植物？僵尸？子弹？

A3：曾想过在存储地图每个块的信息的二维数组的每个对象中，用一个成员表示该项的类型，但在僵尸前进时会出问题（前进之后，原来格子可能有僵尸，可能没僵尸），于是在需要确认状态时，不得不调用 isPlant(pos)等函数判断对应位置是否为植物等（内部遍历了对象列表，对位置进行了比较）。

Q4：庭院地图每个块的绘制问题，效率如何更高？

A4：对于地图每个块，在存储地图每个块的信息的二维数组的每个对象中存在一个 bool 类型的 repaint 成员和 bool 类型的 negative 成员，在绘制地图时，判断 repaint 决定是否重绘该格子内的植物或所有僵尸，判断 negative 决定是否重绘该格子内的子弹。特别的是，绘制僵尸时，先获得该格子内所有僵尸，根据数量决定是罗列还是省略。

五、总结

由于本次课设要求不考虑植物和僵尸共处一格的问题，所以对于地刺和土豆地雷并没有完成实现，此外，因为期中考试的缘故，对于较难实现的舞王僵尸没有完成实现，考虑到舞王僵尸的随机移动和伴舞僵尸的召唤较难实现，另外在控制台里对西瓜射手的抛物线弹道很难模拟，所以只是做了一个简单的直线弹道模拟，将子弹替换为特殊字符，其余在课程要求里的植物和僵尸都完成了实现。

对于最终设计方案的评价，考虑到他们大部分的属性和操作是类似的，所以植物、僵尸、子弹可以均继承一个 base 类，但弊端是造成一定的空间浪费问题，因为有些参数并不是所有的继承类都有的。