第一次课程设计报告

植物大战僵尸①

时间： 3月23日~4月13日

姓名：张玄逸　　学号：201220194

目录

一、概述

1.主要内容

仿照植物大战僵尸，制作一个基于控制台的简易版游戏。玩家需要收集并花费阳光，种植植物以击退试图入侵房屋的僵尸，当有僵尸越过草坪左侧防线时游戏结束。

2.目标

场景：白天（纯草坪无水池迷雾，有阳光掉落）

模式：无尽模式（除非僵尸越过左侧防线，否则游戏不会停止）

规则：每击杀一个僵尸都会获得一定分数，分数累积

植物：向日葵、豌豆射手

僵尸：普通僵尸

3.设计思路

①复习（学习）前置知识：面向对象编程，控制台界面编程

②划分模块：将整个程序划分为 game,store,lawn,plant,zombie,overall 六个版块

其中前五个模块包含对应的类，最后一个 overall 不含类，只含常量和全局函数，常量如窗口大小，颜色，草坪大小，帮助栏宽度，全局函数如隐藏光标，移动光标，按某种颜色输出等。

③确定从属关系：根据空间和逻辑关系来区分主次，确定包含与被包含的关系

④明确功能：五个大类分别执行各自功能（后文会详细阐述）

二、主要类的设计

1.类的数据与操作

①Game 类

Game 类是游戏的总体框架，负责进行整个游戏界面的初始化与更新，僵尸与植物的行动，并且接受玩家的按键并给出相应的反馈。

```
//开始循环
void Game::start() {
    while ( true ) {
        Sleep( single_time );
        if ( state == Pause ) { ... }
        if ( state == Shopping ) { ... }
        if ( state == Planting ) { ... }
        if ( state == Wiping ) { ... }
        if ( state == Pause ) continue;
        store_action( *this );
        lawn_action();
        bullet_action();
        lawn.refresh( *this );
        print_bullet();
    }
}
```

Game 共有四个状态：商店，种植，铲除，暂停。具体操作细节在三、1.运行操作方法中会阐述。

Overall.h 中定义了 single_time=100，代表单位时间。Game 运行的方式是无限循环，而 single_time 被用于 Sleep(single_time)，使得每次循环都会暂停 0.1 秒。将连续的时间分割有利于对植物，僵尸，阳光的

冷却计时，使草坪商店不因频繁刷新而产生闪烁，也保留了足够小的间隔使得肉眼能将画面衔接起来。

```
class Game {
    friend class Store;
    friend class Plant;
    friend class Lawn;
    friend class Bullet;
    friend class Grid;
private:
    Lawn lawn;
    Store store;
    list<Bullet*>bullet_list;//子弹列表
    enum State { Shopping, Planting, Wiping, Pause }state;//状态
    int store_plant_x;
    int store_plant_y;//当前要种植的植物相对坐标
    int x;//当前光标相对位置，与状态有关
    int y;
```

Game 中包含草坪，商店，子弹，状态，植物坐标和光标相对位置。子弹作为一个单独的 class，在 Plant 中只保存了一个表明类型的 int，而当前子弹的完整数据保存在 Game 中而不是 Plant 中的原因是，子弹和植物其实是相对独立的两个部分，植物只需要产生子弹，之后不会对子弹造成任何其它影响。植物消亡后不会再产生新的子弹，但未击中僵尸或碰到边界的子弹仍然处于运行中。

全部成员函数

```
//初始化
Game();
//开始循环
void start();

//商店模式
void shop();
//是否选中商店植物
void plant_selected(int x, int y,int z);
//是否取消种植
void unplant(int z);
//购买植物
bool buy(int z,int w);

//种植模式
bool plant( int z , int w );
//选中草块
void grid_selected(int x, int y,int z);
//取消选中草块
void grid_unselected(int x, int y);

//铲除模式
void wipe();
//是否选中铲除
void shovel_selected( int z );
//是否取消铲除
```

```
//将植物铲除
bool cancel_plant(int x, int y);

//暂停模式
void pause();

//子弹行动
void bullet_action();
//打印子弹
void print_bullet();

//草坪行动
void lawn_action();
//商店行动
void store_action(Game& game);

//失败
void game_over();
```

②Store 类

Store 类主要负责界面下方的版块，从左至右分别是提示栏，植物栏，阳光、分数、铲子栏，帮助栏。

其中，提示栏主要根据玩家某些不合理的操作作出相应的提示。

```
chat [ 0 ] = (char*) "";
chat [ 1 ] = (char*) "这块地已经种植了植物";
chat [ 2 ] = (char*) "这块地没有种植植物";
chat [ 3 ] = (char*) "阳光不足";
chat [ 4 ] = (char*) "植物冷却中";
chat [ 5 ] = (char*) "正在添加中……5月11日开放（如果我还能活到第二次课设的话）";
```

```
class Store {
    friend class Game;
    friend class Plant;
    friend class Grid;
private:
    int sun_cooling;//目前生产单个阳光进度
    int sun_circle;//生产阳光的间隔
    int sun;//阳光总数
    int sun_flash;//阳光闪烁阶段
    int chat_circle;//提示信息显示时长
    int chat_cooling;//提示信息已显示时长
    int now_chat;//当前提示编号
    Plant plant_list [ 3 ] [ 4 ];//可购买的植物
    int score;//分数
    bool score_refresh;//分数是否更新
    char* chat [ 101 ];
```

Store 需要记录提示信息，阳光，植物的冷却情况，在需要更新时重新输出。还需要存储每个格子对应的植物，以便在购买时判定是否冷却完毕以及阳光是否足够。

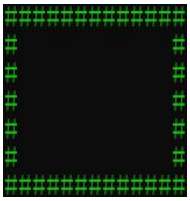
阳光生产时间定为 7.5 秒每个，因为没搜到精确数据，所以这是我掐秒表掐出来的，实际时间在 5.5-9.5s 之间波动，还不是很确定（毕竟一边打僵尸一边掐表太难了），之后会在基本确认具体时间间隔后实现一定的时间波动。

③Grid 类

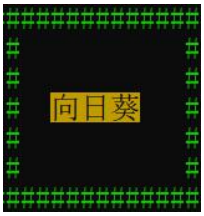
```
class Grid {
    friend class Game;
    friend class Lawn;
    friend class Plant;
    friend class Store;
private:
    int x;
    int y;//绝对位置
    Plant* plant;//该格子上的植物
    list<Zombie*>zombie_list;//该格子上的僵尸
    bool refresh;//是否需要重新绘制
```

Lawn.h 里面有 Grid 和 Lawn 两个类，其中 Grid 是单个草块，被包含在草坪 Lawn 中。

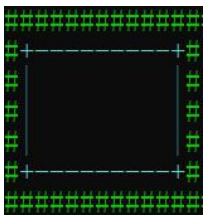
单个草块的大小为 5*11（不包含边界），颜色为深绿色，用‘#’绘制。

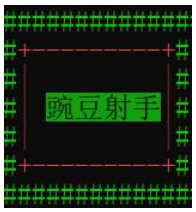


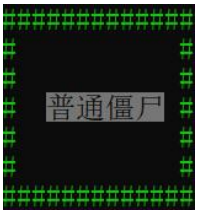
有植物时

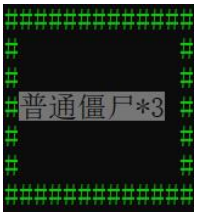


种植模式被选中时　　铲除模式被选中时　　有僵尸时









僵尸与植物同存时

全部成员函数

```
//初始化
void init( int x , int y );
//打印植物，僵尸，选择框
void print( Game& game );
//获取植物颜色
int plant_color();
//植物行动
void plant_action( Game& game );
//向日葵行动
void sunflower_action( Game& game );
```

```
void sunflower_action( Game& game );
//射手行动（包括豌豆，寒冰，双发三发，机枪）
void shooter_action( Game& game );
//判断豌豆攻击范围内是否有僵尸
bool pea_can_attack( Lawn& lawn );
//僵尸减血
void zombie_minus( int x , Game& game );
//判断僵尸是否能攻击植物
bool zombie_can_attack();
//植物减血
void plant_minus( int x );
```

④Lawn 类

```
class Lawn {
    friend class Game;
    friend class Plant;
    friend class Grid;
private:
    int zombie_cooling;//生产单个僵尸的进度
    int zombie_circle [ 21 ] [ 2 ];//僵尸产生的时间间隔与数目
    int whole_wave;//完整的一轮包含的时间间隔数目
    Grid grid_list [ 9 ] [ 5 ];
```

Lawn 只包含 5*9 个草块和一些关于生产僵尸的常量，草坪的作用主要就是遍历实施单个 Grid 的操作和生产僵尸。在 Zombie 类会详细解释 zombie_circle 与 whole_wave 的含义。

草块的存储是先列后行，因为<windows.h>中设置光标的函数参数为先列后行，为避免混乱就统一采用此种顺序。

⑤Plant 类

属性 / 种类	名称	类型	防御力	价格	冷却时间(s)	特殊属性
向日葵	sunflower	0	300	50	7.5	第一次隔 7.5 秒生产阳光，之后每隔 24 秒生产一次
豌豆射手	peashooter	1	300	100	7.5	每 1.4 秒发射一颗豌豆子弹

注 1：红字数据为掐秒表掐出来的，并非百科数据，可能有些许差别。

注 2：豌豆射手刚种下时貌似有一段时间的缓冲，也就是即使前方有僵尸，也不会立即发射子弹。经测量，平均缓冲时间约为 1 秒。

```
//攻击类植物特性
int attack_cooling;//当前已冷却的攻击时间
int attack_time;//攻击间隔时间
//时间间隔单位均为0.1秒
int bullet_type;
//子弹种类

//向日葵特性
int make_sun;//生产阳光数
int start_sun_circle;//向日葵第一轮阳光生产时间
bool if_start;//是否已进行第一轮生产
int sun_circle;//生产阳光间隔
int sun_cooling;//目前阳光冷却时间
```

部分类的数据成员和成员函数中含有仍未实现的功能，比如射手添加不同的子弹。因为第一次只要求实现豌豆射手，所以其它类型的植物和子弹留到了下次。

目前实现的两种植物没有采用子类继承，因为植物种类数目较少（而且我还没学继承）。

⑥Zombie 类

属性 / 种类	名称	类型	防御力	攻击力	速度	击杀分数
普通僵尸	normal	0	200	10	4.7 秒一格	5

注 1：攻击力指的是一个单位时间内对单个植物产生的伤害，即 0.1 秒内对植物造成的减血

注 2：“一格”指的是一个草块，也就是（单个草块宽度+1=12）个字符的宽度

```
zombie_circle [ 0 ] [ 0 ] = 0;
zombie_circle [ 1 ] [ 0 ] = 200;
zombie_circle [ 2 ] [ 0 ] = 500;
zombie_circle [ 3 ] [ 0 ] = 770;
zombie_circle [ 4 ] [ 0 ] = 1070;
zombie_circle [ 5 ] [ 0 ] = 1300;
zombie_circle [ 6 ] [ 0 ] = 1600;
zombie_circle [ 7 ] [ 0 ] = 1700;
zombie_circle [ 8 ] [ 0 ] = 1830;
zombie_circle [ 9 ] [ 0 ] = 1950;
zombie_circle [ 10 ] [ 0 ] = 2050;
zombie_circle [ 11 ] [ 0 ] = 2200;
zombie_circle [ 12 ] [ 0 ] = 3200;
zombie_circle [ 13 ] [ 0 ] = 4100;
zombie_circle [ 14 ] [ 0 ] = 5000;
```

```
zombie_circle [ 0 ] [ 1 ] = 1;
zombie_circle [ 1 ] [ 1 ] = 1;
zombie_circle [ 2 ] [ 1 ] = 1;
zombie_circle [ 3 ] [ 1 ] = 1;
zombie_circle [ 4 ] [ 1 ] = 2;
zombie_circle [ 5 ] [ 1 ] = 2;
zombie_circle [ 6 ] [ 1 ] = 3;
zombie_circle [ 7 ] [ 1 ] = 3;
zombie_circle [ 8 ] [ 1 ] = 3;
zombie_circle [ 9 ] [ 1 ] = 4;
zombie_circle [ 10 ] [ 1 ] = 4;
zombie_circle [ 11 ] [ 1 ] = 4;
zombie_circle [ 12 ] [ 1 ] = 6;
zombie_circle [ 13 ] [ 1 ] = 6;
zombie_circle [ 14 ] [ 1 ] = 9;
```

在游戏的最开始，僵尸的出场有固定的时间间隔。因为刚开始收集阳光，植物未形成有效的防御线，加上键盘操作增加了困难度，且植物栏未完善，所以不可能按照无尽模式 6 秒一波的速度来产生僵尸。于是我又双叒叕掐了表。

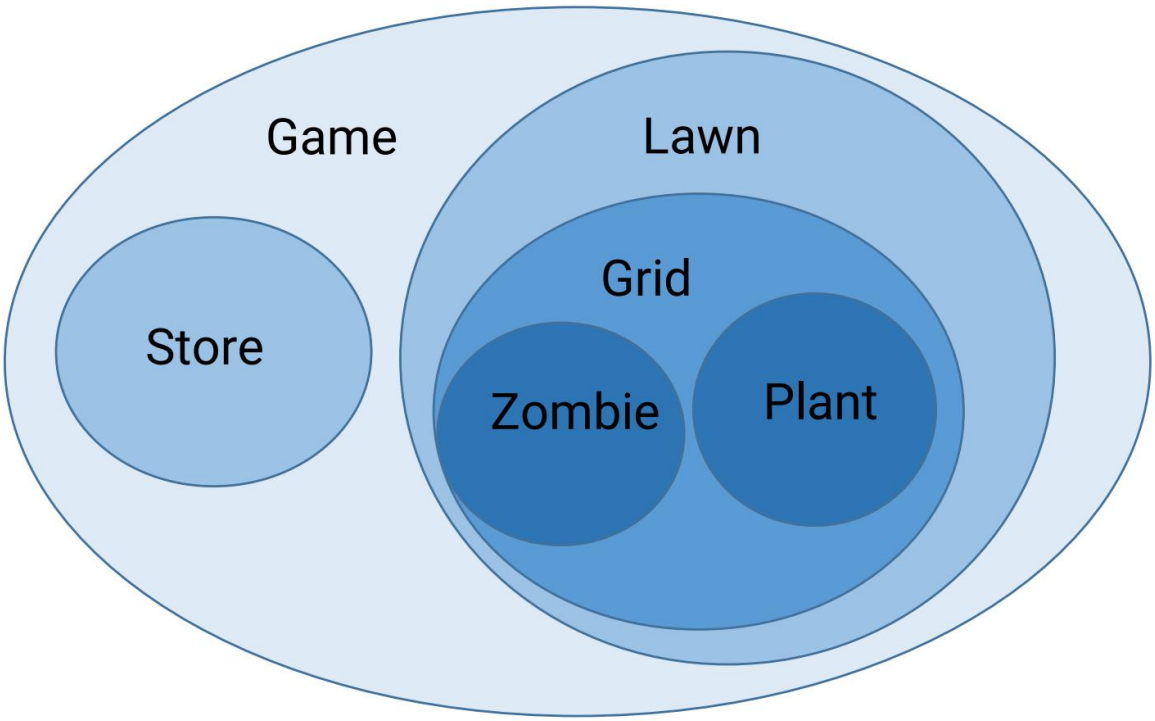
上图是按照挑战模式 1-1 关卡的时间间隔得出的，在第一个摇旗僵尸出现前后的僵尸时间数量分布。其中 1-14 行代表第 1-14 波僵尸的数据，第 0 列为从游戏开始到僵尸出现的总时间（以 0.1 秒计），第 1 列为该波僵尸数目。比如，zombie_circle[7][0]=1700,zombie_circle[7][1]=3，代表在第 7 波在第 170 秒，有 3 个僵尸出现。而 zombie_circle[0][0]无用，zombie_circle[0][1]用来记录当前正在靠近的时间线，比如现在过了 530 个单位时间，正在准备产生第 3 波僵尸，那么 zombie_circle[0][1]=3。

注：数据有波动，这里只选取了大致的时间间隔

whole_wave 用于记录总的时间间隔数目，即 14，用于判断是否已完成最初的循环。如果是，则当前时间跳回第 12 波，相当于接下来永远循环第 12-14 波的产生。

2.类之间的关系

6 个类的关系大致如下。



三、程序运行

1.运行操作方法

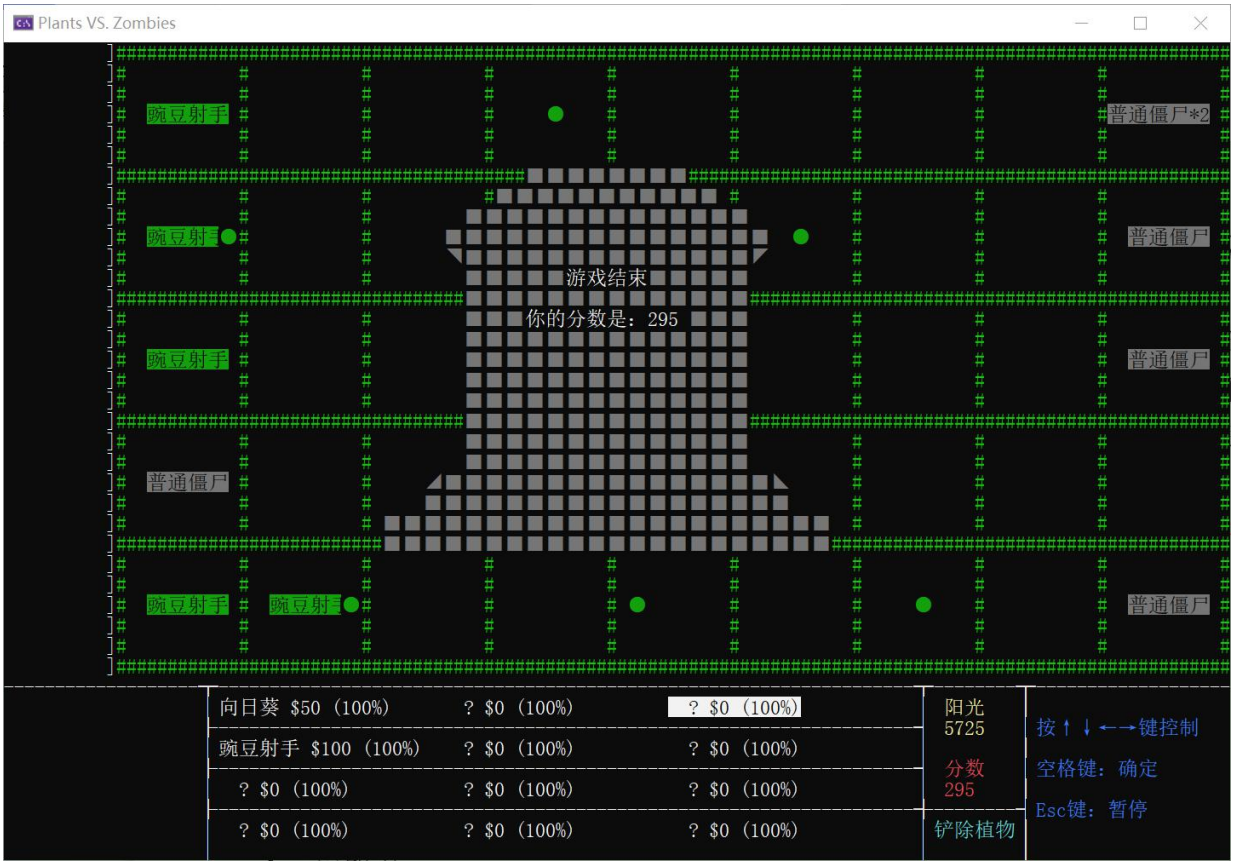
游戏主界面如图，主要部分是 5*9 的草坪，左侧为房屋的边界线。

商店：此为游戏的默认状态，光标（此处光标不是控制台光标，而是指当前选择的草块，植物，铲子等相对位置）可以在植物栏中移动，空格键确认后若满足购买条件则进入种植模式。此次只有向日葵和豌豆射手可以购买。

种植：光标可以在草坪间移动，种植成功则返回商店模式。购买状态下，选择的植物栏位置自动变为“取消种植”按钮，若想取消，则只需按↓键将光标移动到此按钮上，确认即可。

铲除：与种植类似。特别地，在植物栏最右一列按→键可选中铲子，确认即可进入铲除状态。

暂停：按 esc 键使游戏暂停，直到按下空格键才开始正常循环。注：只有在商店状态下才能暂停，也就是必须归还铲子或者种植完毕后按 esc 键才会起效。



此为游戏结束界面。（我正在想办法让这个墓碑看起来像一点 QAQ）

注：因为操作比原游戏要麻烦一点，所以植物的选择与种植可能会花费一点时间。如果打不过就找到 store.cpp，将第一个构造函数中的 sun 改多些

2.功能亮点

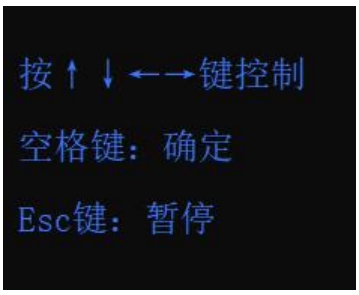
①操作方式

关于植物购买、地块选择和植物种植，参考流程是

1. 捕捉键盘字母 b 表示购买某种植物
2. 焦点移动至庭院，通过上下左右键选择空的地块

3. 捕捉回车键确认种植，字母 x 取消

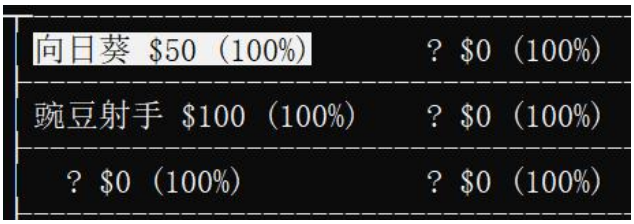
但我认为此种方法需要用到的按键太多，难以记忆（特别是字母键），会导致操作上面浪费大量时间，对于这类时间掌控度要求高的游戏来说就可能导致失败。因此我在整个游戏流程中只设置了 6 个有效按键，分别是上下左右键，空格键，esc 键，完全抛弃了字母键的使用。上下左右键用于光标的移动，空格键用于确认，esc 键用于暂停。这些按键的功能从逻辑上都很好理解，操作起来也比较方便。



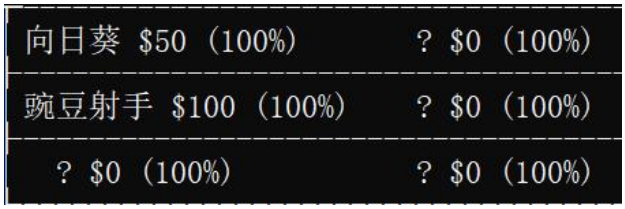
②视觉效果

为了让玩家能清楚地知道当前光标的位置，我设置了多个 selected 和 unselected 函数，用于打印草块，植物栏，铲子。

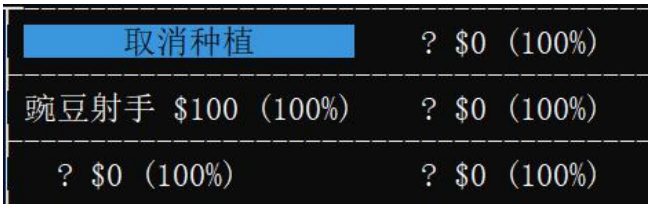
A 光标停留的植物栏（未按空格键确认）



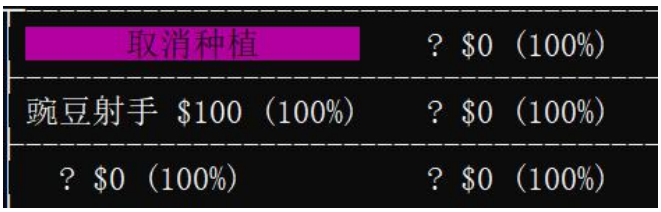
B 未停留的植物栏



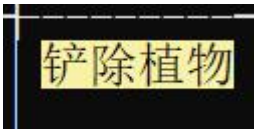
C 确认购买后（未种植）



D 光标停留时（未按空格键确认）



E 光标停留的铲子

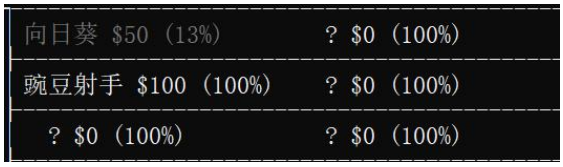


F 未停留的铲子

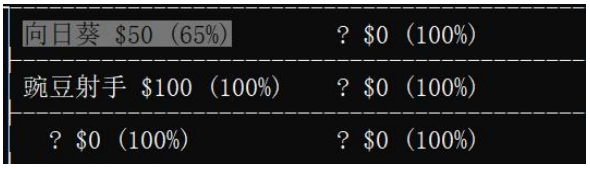
G 选中铲子后（未铲除）

H 光标停留时

I 冷却时



J 光标停留时



AB,CD,EF,GH,IJ 是光标是否停留的不同表现；在 A 处和 E 处按空格键后跳转到 C 和 G，光标移动到草坪左下角；在 D 和 H 处按空格键后跳转到 A。

③各类数据的设置

草坪：采用标准的 5*9 格，缺点在于还没有实现像真正的植物大战僵尸那样，在最右侧多出可以看到僵尸出现但植物还攻击不到的时间段。

自然阳光间隔，向日葵阳光间隔，初始僵尸循环的时间数目：均为掐秒表得出，不一定十分准确，但尽可能保持接近。

植物，僵尸的攻击力，防御力，冷却时间等数据：查询搜狗百科而得。

子弹速度：未查询到，主要依靠暂停时观测两颗子弹的距离得出。

总之，为尽可能接近游戏的原本体验，我尽可能地获取了标准的数据。

四、遇到的问题与解决方案

char 类型的实参与 LPCWSTR 类型的形参类型不兼容

解决方案：

①项目->XXX 属性->配置属性->高级->字符集，由使用 Unicode 字符集改为使用多字节字符集

②#define char TCHAR

Visual Studio 2017 中“const char *”类型的值不能用于初始化 “char *” 类型的实体

解决方案：项目属性->C/C++->语言>符合模式项>选择否

list 的遍历删除

解决方案：

```
for(list<int>::iterator i=team.begin();i!=team.end();){
    if(f(*i)) team.erase(i++);
    else i++;
}
```

int 转化为 string

解决方案：#include<string> to_string(x);

等待玩家输入特定的按键并不显示在屏幕上
解决方案：

```
#include<conio.h>
while(true){
    if(_kbhit()){
        char c=_getch();
    }
}
```

头文件包含问题
解决方法：
①只声明 class A
②用指针
③尽量在.cpp 文件里包含头文件

子弹有时会径直穿过僵尸而不是消失
解决方案：我发现是因为每个循环都有一次僵尸移动，子弹移动，而子弹移动完毕后才判断该格是否有僵尸。如果在第 x 秒，子弹在僵尸左边一格，第 x+1 秒子弹右移，僵尸又刚好左移的话就会错开。之后我添加了子弹在移动前和移动后的两次判定。

五、总结与反思

程序的不足之处在于，各个模块的功能其实并不是很明确地被区分开来。因为有包含与被包含的关系，所以下一层的类的操作也能由上一层直接操作来实现，比如 game 包含 lawn,lawn 包含 grid，则 game 可以直接处理 grid，这样就导致我有时并不是很明确应该将具体的操作交给哪一层，模块功能比较混乱，在我的程序中大概见不到 Demeter 法则的影子。

另外，因为我太弱，在课设开始的两个周之前连.h 和.cpp 都不会用，更不知控制台为何物，一上来就写简易版植物大战僵尸，未免有些手忙脚乱，一边学控制台界面编程一边努力理清.h 和.cpp 的关系。在写的过程中，由于严重缺乏面向对象编程的经验，我的程序经常需要修补。所以整个程序代码并没有呈现出一个很清晰的框架，基本上是走一步算一步。

回过头发现，其实控制台这部分确实不难，只不过一开始由于没有了解过，加上众多复杂的函数才让我心生恐惧。事实上写一个简易版的植物大战僵尸用不到什么特别高深的代码知

识，上图众多版块的大量函数中其实只用到了设置窗口，光标，文字颜色三类操作，多余的根本派不上用场。

```
//设置窗口标题
void set_screen_title(TCHAR* x);
//设置窗口大小
void set_screen_size();
//设置颜色
void set_color(int color);
//带颜色输出
void print_in_color(const string& x, int color = default_color);
//设置光标位置
void set_cursor(int x, int y);
//隐藏光标
void hide_cursor();
```

关于控制台的操作函数都放在了overall.h里面，左图即为overall.h包含的全部函数。

一旦完成了最基础的全局函数，剩下的工作与windows.h就没什么关系了。整个project考察的主要就是面向对象的思维方式，控制台的确只是一个实现的工具。代码的实现需要的更多是细心与耐心，以及统筹规划的能力，而不是高级的算法和数据结构。

这次课设让我想起了上学期的SICP，两门选修课的共同点就是可以把人逼疯的project。当然，这也的确让我受益匪浅。正是高难度的课程逼迫我发奋学习相关知识点，并且通过写大量的实践练习发掘容易忽视的细节，掌握不熟悉的方法，改正错误的习惯，从而提升自己的编程水平。说实话，如果植物大战僵尸这个项目没有ddl，那写起来真是一种休闲娱乐。

谢谢！