《程序设计实践A》
课程设计报告

学生姓名：苏一涵

学　　号：36720232204041

专业班级：软件工程

教　　师：洪青阳

二〇二四 年 七 月 二十 日

目　　录

1. 课程设计选题内容与分析

1.1 选题内容

我的选题是“用 C 语言复现植物大战僵尸”，希望通过个人独自完成经典游戏 pvz 的复刻来巩固小学期学习的 C 语言知识，同时提升自己的编程与项目规划能力。

1.2 分析

植物大战僵尸作为一款经典的用 C++编写的 PC 游戏，在网上有极高的人气，因而有许多用不同语言复现的 pvz 版本在网络上可以参考，同时作为一个十几年前的游戏，在今天想要大致的复现难度并不是很大。所以作为个人项目在两个星期到三个星期完成原始开发是可以达成的。最后，用 C 语言来复现可以加深我对底层开发的认识以及个人的 C 语言能力。

2. 设计方案

2.1 项目描述

《植物大战僵尸》是一款极富策略性的小游戏。可怕的僵尸即将入侵，每种僵尸都有不同的特点，例如铁桶僵尸拥有极强的抗击打能力，矿工僵尸可以挖地道绕过种植在土壤表面的植物等。玩家防御僵尸的方式就是栽种植物。植物每种都有不同的功能，例如樱桃炸弹可以和周围一定范围内的所有僵尸同归于尽，而食人花可以吃掉最靠近自己的一只僵尸。玩家可以针对不同僵尸的弱点来合理地种植植物，这也是胜利的诀窍。

《植物大战僵尸》集成了即时战略、塔防御战和卡片收集等要素，玩家控制植物抵御僵尸的进攻，保护这片植物园。游戏中可以选用的植物有 40 多种，而每个场景最多只能选用 8 种植物，这就需要玩家根据自己的游戏策略来作出取舍。

这款游戏要求玩家既有大脑的智慧，又要有小脑的反应。在有了正确的战略思想之后，还要靠战术将战略实现出来。战术范围包括很广，植物的搭配、战斗时的阵型、植物与僵尸相遇时，是战是防这都属于战术的范畴。正确的战术是玩家在战斗中胜利的关键，选择正确的战术，需要先分析情况，再做出决定。那么提高战术水平也是要提高分析情况的能力。

每个植物的安放需要消耗阳光，且植物安放完后还需要冷却完成后才能再次安放，游戏胜利的条件是消灭所有僵尸，如果让僵尸进入房子则游戏失败。

2.2 要求

该游戏启动后，游戏将初始化，显示出 900*600 的游戏窗口，同时进入主菜单页面；游戏应该能够读取用户的鼠标输入操作，在用户点击开始游戏按钮后进入攻防界面，游戏开始；应该有开场动画；包含植物，僵尸，阳光以及对应的背景音乐；要有游戏结束的判定，同时要给出游戏胜利或失败的动画；植物和僵尸要有对应的不同功能。

基于上述生成的地图，游戏根据用户的操作实现 pvz 游戏效果；

游戏应有可视化用户界面；

2.3 方案

依照我个人开发的习惯将项目分为游戏场景创建，添加元素，画面渲染，功能实现，程序停止条件五个部分。

依次实现不同的模块，最后得到完整的游戏。

2.3.1 游戏场景创建（主循环）

开始界面，对战场景以及植物栏，使用 easyX 图形库函数把图片在指定位置加载器出来即可(IMAG：图片类型函数；loadimage：载入图片；putimage：渲染图片)

2.3.2 添加元素

包括植物，僵尸，阳光，背景音乐等元素。通过定义各自的结构体来实现其对应的各种功能，音乐直接通过 playsound 函数来实现。

```
struct plants {//实现植物
    int type;//植物类型，0
    int frameIndex;//帧序
    bool catched;//是否正
    int deadTime;//植物血
    int status; //植物状态
    bool shovelAndCatched;
    int prepareTime;//准备
    bool boom;  //是否爆炸
    int x, y;   //植物二维
};
struct plants map[5][9];//
```

```
struct sunshineBalls {//阳光的
    int x;//x不变
    int y;//阳光坐标
    int frameIndex;//正在播放的
    int destY;//飘落位置的纵坐
    int used;//是否在使用
    int timing;//阳光存在时间
    float xoff;//阳光收集时的移
    float yoff;
    bool flower;        //是否
    bool click;         //是否
};
struct sunshineBalls balls[10];
int sunshine;//可用阳光数
```

```
struct zm {//定义僵尸结构
    int x, y;//位置
    int frameIndex;//播放
    int used;//是否使用
    int speed;//僵尸速度
    int row;//僵尸在哪一
    int blood;//僵尸血量
    bool dead;//死了没
    bool eating;//僵尸吃
    int type;//僵尸种类
};
struct zm zms[20];//一关
```

2.3.3 画面渲染

用一个死循环（游戏主循环）来不断渲染游戏主场景；对于僵尸，植物，阳光的渲染，由于不能直接播放 GIF，所以采用一帧一帧渲染来实现。这里用一个二重

循环来遍历所有的植物，渲染对应的帧图片，依托游戏主循环不断播放实现动图的效果。

```
int main() {
    gameInit();//游戏初始化
    settextType();
    startUI();//开始菜单
    int time = 0;
    bool flag = true;            //设置延迟
    while (1) {
        userClick();//输入
        readySetPlant();
        updateWindow();//更新窗口
        time += getDelay();
        if (time > 30) {
            flag = true;
            time = 0;
        }
        if (flag) {
            updateGame();//更新数据
            updatedraw();//渲染
            checkGameOver();
            flag = false;
        }
    }
```

游戏主循环

渲染函数

```
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 9; j++)
    {
        if (map[i][j].type > 0) {
            int x = curX00 + j * cur_Width;
            int y = curY00 + i * cur_Height;
            int plantsType = map[i][j].type - 1;
            int index = map[i][j].frameIndex;
            putimagePNG(x, y, imgPlants[plantsType][index]);
```

2.3.4 具体功能实现

用户操作：由于植物大战僵尸的所有操作都可以用鼠标实现，这里用 easyX 库的 ExMessage 结构体来存储鼠标输入，用 peekmessage 来读取鼠标数据，再用 Window.h 库函数来判断具体操作，通过读取鼠标的位置坐标，可以实现各种操作功能。

植物僵尸功能：以僵尸为例，僵尸除了移动还要可以吃植物，我这里用僵尸与植物之间的坐标差来作为判断依据，当僵尸与植物坐标差在 20 个像素之内，开始渲染僵尸吃植物的动画，同时植物死亡值累加，这边渲染操作同上。

```
void checkZomToPlants() {                    //僵尸吃植物的碰撞检测
    for (int i = 0; i < zmMAX; i++) {
        if (zms[i].used) {
            for (int k = 0; k < 9; k++) {            //这一行的九个草坪块
                int row = zms[i].row;
                if (map[row][k].type == 0)
                    continue;                        //如果这个草坪块没有植物就略过
                int plantX = 256 - 120 + k * 81;
                int x1 = plantX + 110;               //x1和x2是植物的边界
                int x2 = plantX + 160;
                int x3 = zms[i].x + 80;              //x3是僵尸的左边界
                static int count[zm_BeforeWave] = { 0 };
```

2.3.5 程序停止条件

当所有僵尸都死亡或者僵尸进入房间（离背景左侧小于 16 个像素）,赋予 gameStatus 胜利或者失败的变量（WIN==1，FAIL==2），当检测到 gameStatus 状态改变，程序停止（exit），播放胜利&失败动画。

```
void checkGameOver() {
    if (gameStatus == FAIL) {//僵尸吃掉了你的脑子
        putimagePNG(220, 80, &imgFail);
        mciSendString("close bg2", 0, 0, 0);
        mciSendString("play res/audio/lose.mp3", 0, 0, 0);
        MessageBox(NULL, "Game Over!", "over", 0);  //游戏结束（还能优化）
        exit(0);
    }
    else if (gameStatus == WIN) {//播放胜利动画

        mciSendString("close bg2", 0, 0, 0);
        mciSendString("play res/audio/win.mp3", 0, 0, 0);
        for (int i = 1; i <= 45;i++) {
            putimagePNG(250, 200, &imgWin[i]);
            Sleep(120);
        }
        MessageBox(NULL, "You are victorious!", "over", 0);
        exit(0);
    }
```

3. 问题及解决方案

3.1 植物僵尸等的动画快速抽搐

分析：动画的渲染是在游戏主循环中进行的，动画快速抽搐是由于图片帧的渲染过快，导致看起来像是在抽搐。

解决方法：给渲染函数添加一个延迟函数，让它几十个循环才渲染一次，这样动画的播放速度就正常了，即设置一个变量 time，每一次循环 time++，当 time%30==0 时执行一次渲染。

3.2 阳光归位位置会有较大偏移

在点击阳光后，阳光会飞到植物栏左侧，但是经常会飞到旁边的位置。

分析：用一个三角函数来规定回归路线，但是阳光坐标一直在改变，误差会积累，导致返回位置有极大偏差。

解决方法：阳光每移动一次，都重新计算它与指定位置的正切值，修正路径，误差减少到最低。

3.3 阳光收集音效无法播放

经过调试发现，阳光收集音效无法播放或者只能播放一小段。

分析：网上查找发现 mciSendString 函数播放 MP3 文件时，如果已经有音乐在播放会出现播放失败的情况。

解决方法：改用更新的 PlaySound 函数来播放音乐。

3.4 僵尸动画播放异常

当新的僵尸生成时，前面所有僵尸的动画停止播放，同时新僵尸的动画播放速度变为僵尸数的倍数。

分析：动画渲染函数正常运行，同时植物与阳光的动画播放正常，推测是僵尸数据更新函数出现问题。

解决方法：调试发现遍历僵尸的二重循环用的同一个变量，更改后再把僵尸数据的更新从 drawzombie 中独立出来到 updatazombie 增强代码的可读性。

3.5 不同种类僵尸渲染异常

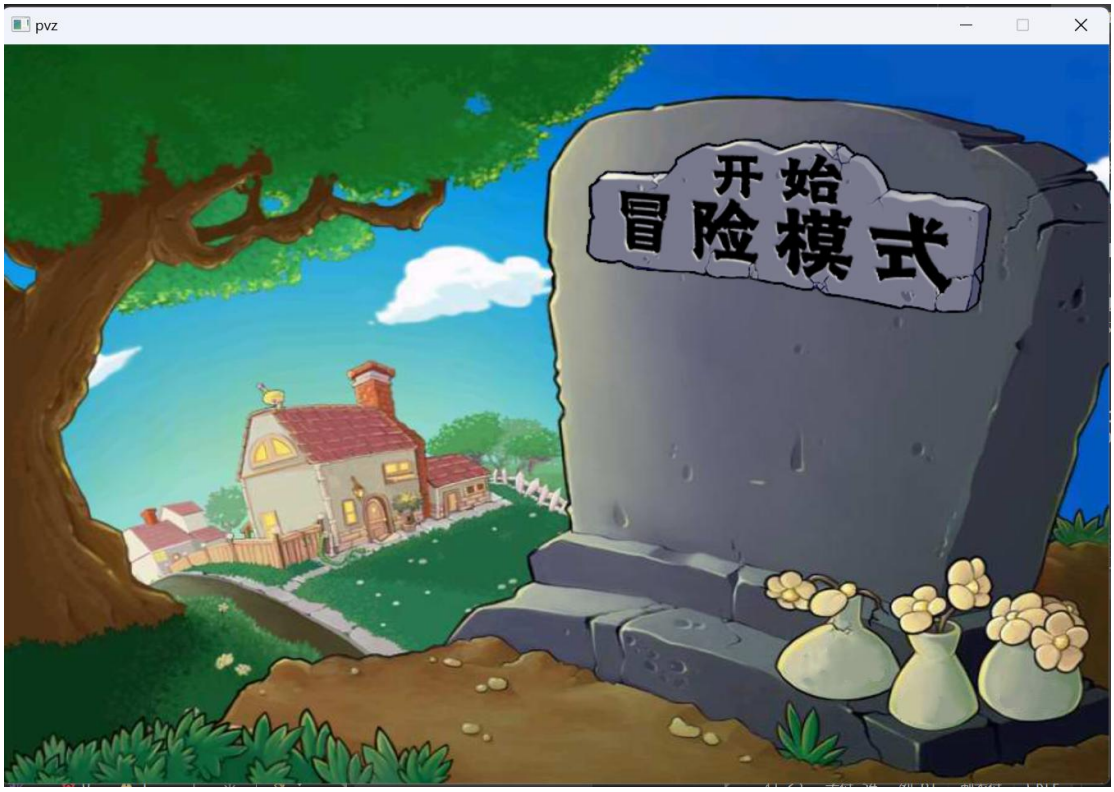
运行发现只能生成普通僵尸，且僵尸在吃植物时又会一瞬间变成路障僵尸或者铁桶僵尸。

分析：通过运行调试发现，僵尸的血量是有不同的，即不同种类的僵尸实际上已经生成了，但是游戏渲染的依旧是普通僵尸的动画帧。

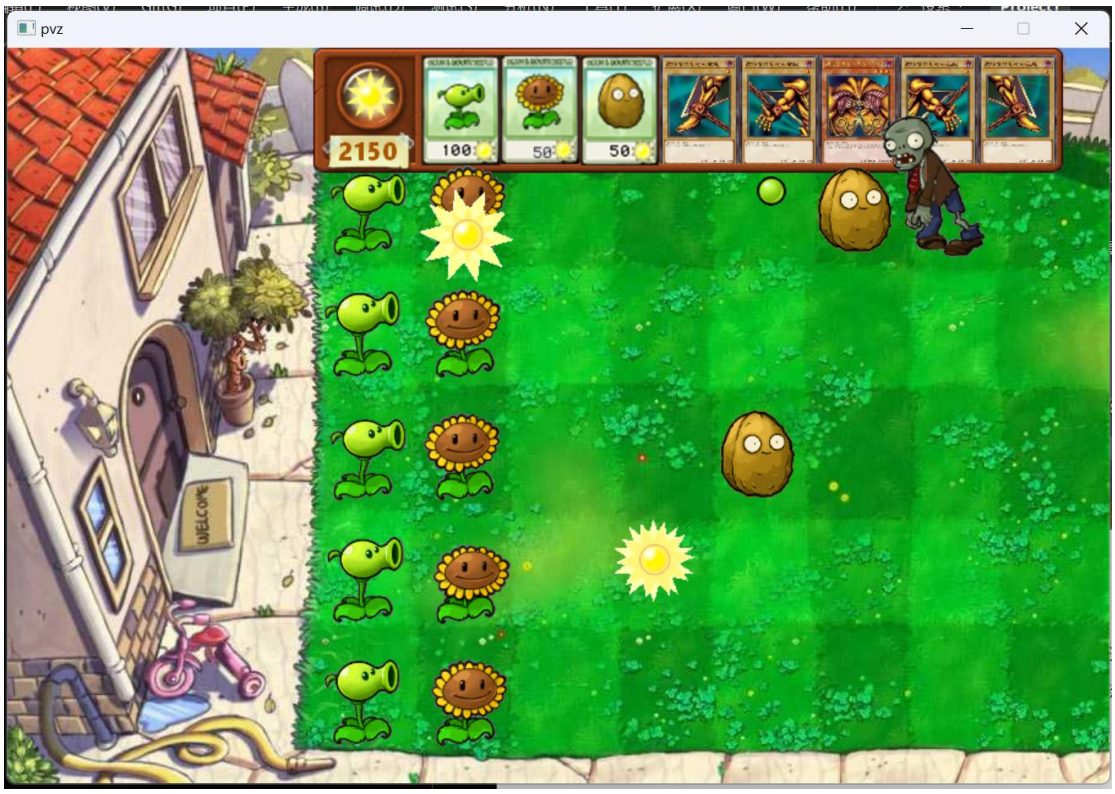
解决方案：通过不断调试，把 drawZombie 函数中的三个僵尸渲染分别独立渲染，可以运行，但是整合后又会出 bug，应该还有更好的优化方案。

4. 实验结果展示

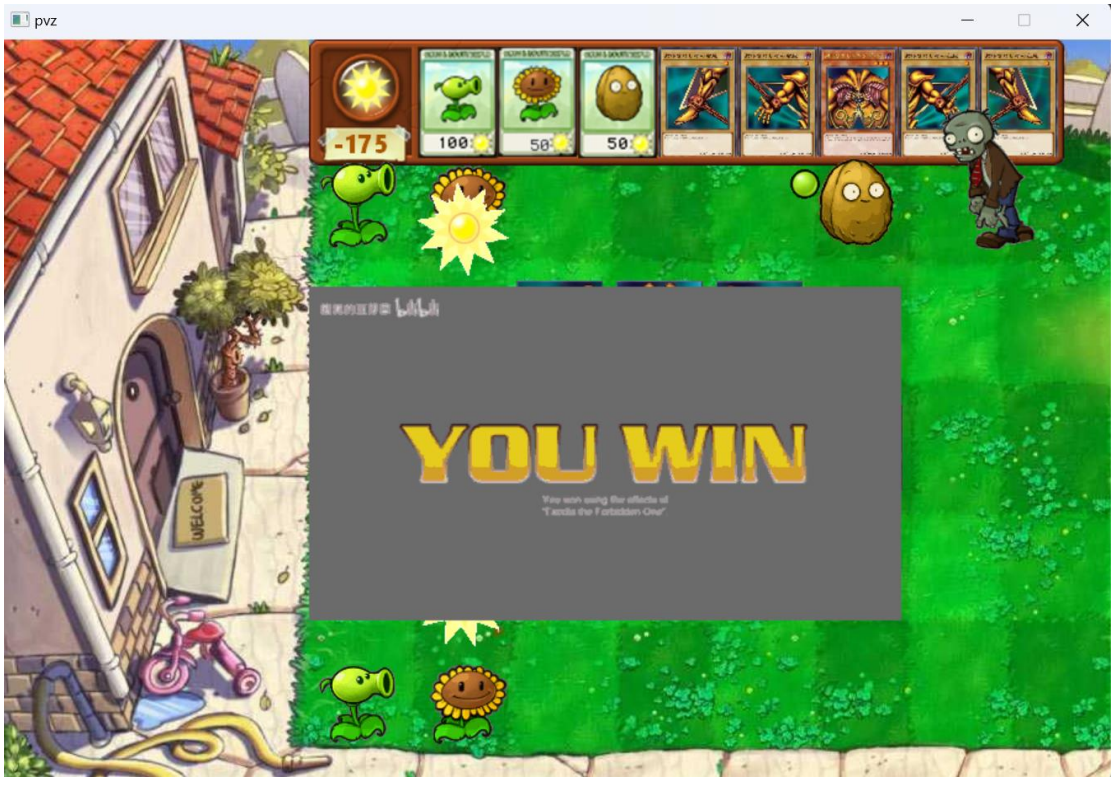
4.1 主菜单

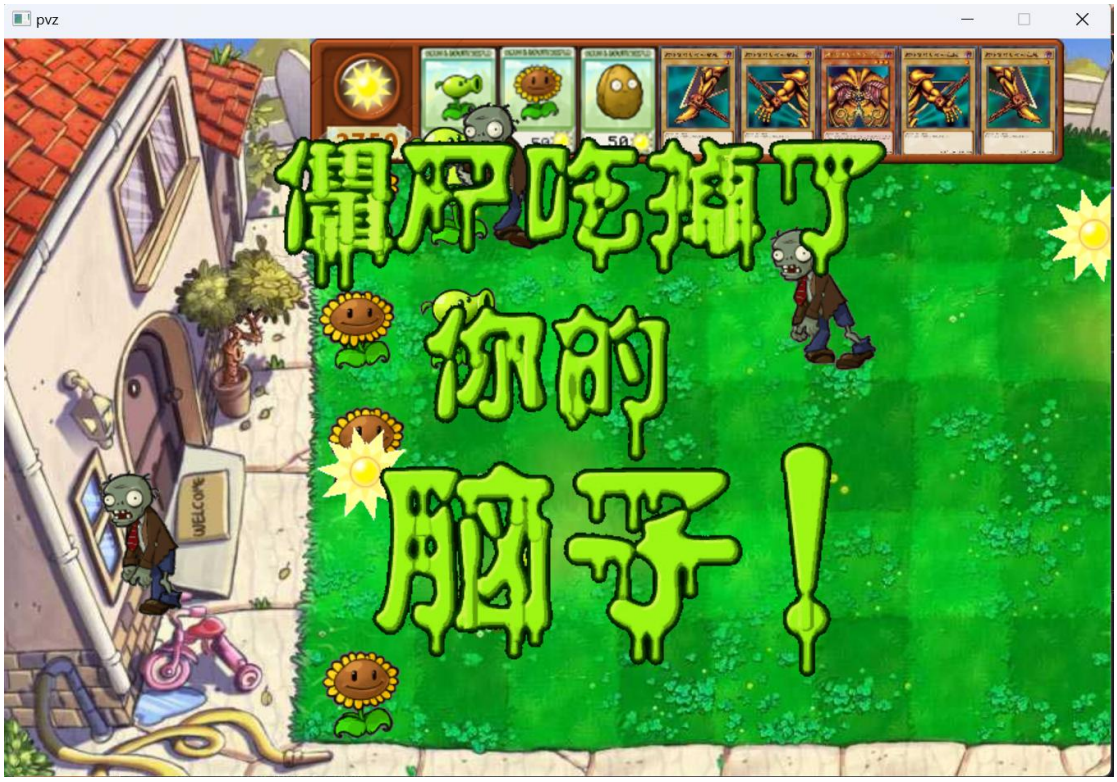


4.2 游戏画面



4.3 游戏胜利和失败画面





5. 本课程学习总结

基础的重要性：通过两周多的 oj 题目练习，让我深感代码编写及问题解决能力的不足，也明白了良好的基础很重要。

理论与实践的结合：将理论知识应用于实际问题的解决中，让我对动态规划有了更深的理解。这种从理论到实践的转化，不仅加深了我的理解，也提高了我的编程能力。

代码质量与效率：在编写代码的过程中，我学会了如何写出清晰、高效的代码。特别是在处理二维数组时，如何合理地使用循环和条件判断，以及如何避免不必要的计算，这些都对代码的效率有着直接的影响。

模块化设计：将一个大问题分解为若干个小问题，每个小问题用一个模块来解决，这样的设计思路让整个项目更加清晰，也便于调试和维护。

测试的重要性：在开发过程中，我不断地测试各个模块的功能，确保每个部分都能正确工作。这让我意识到了测试的重要性，它是保证软件质量不可或缺的一环。

问题解决的思维方式：这个项目锻炼了我的逻辑思维和问题解决能力。面对复杂的问题，如何将其分解、如何找到合适的解决策略，这些都是我在这过程中学到的。

持续学习和改进：在完成项目的过程中，我遇到了很多挑战，但通过不断地学习

和尝试，我最终克服了这些困难。这让我认识到，持续学习和改进是成为一名优秀程序员的关键。