武汉大学国家网络安全学院

《程序设计实验》报告

实验项目名称 飞行射击类游戏的实现方法

姓名 _____ 学号 _

实验学期 2021-2022 学年 第 1 学期

课堂时数_____课外时数_____

填写时间 2021 年 12 月 12 日

实验概述 飞行射击类游戏是经典的小游戏，由耳熟能详的《雷霆战机》到设定庞大的《东方 project》系列，游戏过程十分简单但在实际编写设计思路具有一定难度，在基本实现的基础上进行换皮再加入个人想要达到的功能。
实验项目名称： 飞行射击游戏的实现方法。
实验目的： 1）掌握C程序的语法、程序设计的基本知识与基本技能；

2）提高分析问题和解决问题的实际能力；

3）掌握结构化程序设计的基本思想，重点学习如何建立良好的安全编程思维模式；

4）设计并实现特殊题材的飞机大战游戏。

【实验环境】（使用的软件）：visual studio 2022

(1) 硬件环境：AMD R9 5900HX

(2) 操作系统环境：Windows 10

(3) 编程语言：C/C++

(4) 其他环境：easyx 图形库

【参考资料】：《c 语言项目从零开始编译飞机大战》

一、实验内容设计

1）问题描述与分析

实现飞行射击游戏。

游戏规则：玩家操纵自己的单位发射子弹攻击敌人，并躲避敌人的攻击。

分析：用户只需要输入 w s a d space 即可，对于用户而言操作简单易上手。

首先确定用结构来储存敌我单位的数据，然后要用恰当的函数实现各种操作。这是游戏进行的核心步骤。

2）模块划分

数据初始化；打印画面；游戏控制与判定；判断是否结束；重复循环。

3）实验方案设计：

飞行射击游戏需要一个长方形背景图和游戏中的各种单位，因此使用 easyx 图形库进行图形设计。通过一群不同的函数分别控制不同的功能来达到化繁为简的目的。程序主体分为以下几个模块：主体定义模块，功能函数模块，主函数模块。

首先引用需要使用的各种头文件：

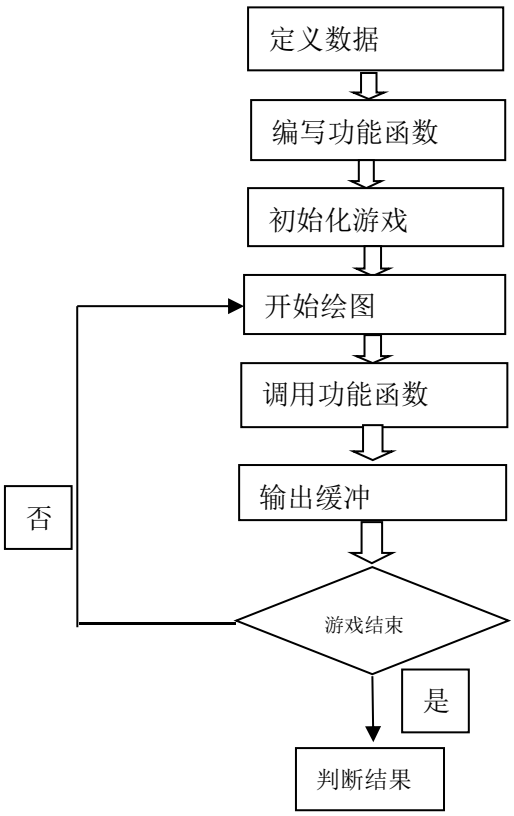
```
#include<stdio.h>
#include<stdlib.h>                    //包含system函数
#include<graphics.h>                  //使用easyx来制作图形窗口
#include<windows.h>                   //包含键盘检测函数
#include<time.h>                      //包含计时器所需要的函数
#include<mmsystem.h>                  //包含声音播放函数mciSendString
#pragma comment(lib,"Winmm.lib")      //包含所需声音播放函数mciSendString
```

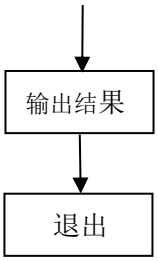
然后使用枚举来定义各种数据达到#define 的效果，使用结构体来定义单位属性。

主函数：

设置简单的开始流程并初始化，使用一个循环来反复调用各个函数实现动态游戏流程：先初始化游戏，然后打印界面，利用 while 形式不断调用功能函数，同时判断游戏是否结束，当游戏结束后告知游戏结果和最终分数。

流程图：





二、核心代码解读：

1）主函数：

```
int main()
{
    printf("欢迎来到VANd大战比利王\n");
    printf("wasd操作移动，空格射击，您的生命值只有3\n当杀敌数达到一定程度时，将会召唤boss，击败boss，取得游戏的胜利吧！ \n");
    system("pause");            //暂停后查看说明，按任意键开始
    gameinit();                 //游戏初始化
    initgraph(WIDTH, HEIGHT);   //生成窗口
    BeginBatchDraw();           //开始绘图
```

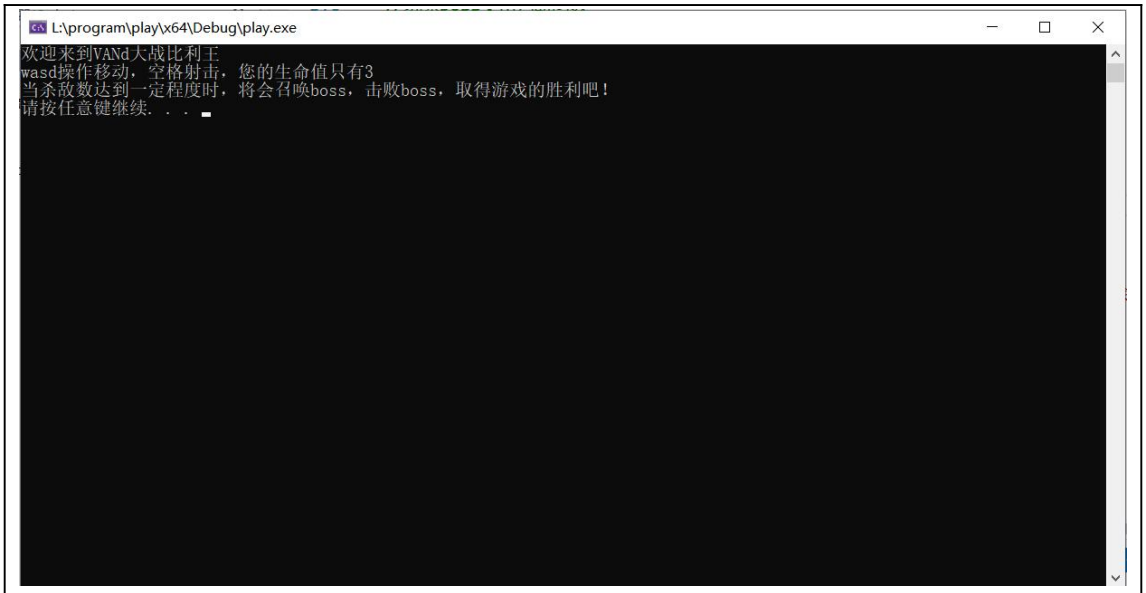
```
while (1)                       //游戏开始
{

    gamedraw();                 //即时绘图
    playermove(0.05);           //控制函数
    bullmove();
    if (timer(500))             //控制敌人生成时间
    {
        create_enemy();
    }
    enemymove(0.02);
    bossmove(0.02);
    hit();
    bosshit();
    bossbullmove();
```

```
        if (player.live == false)    //游戏结束条件判断，玩家死亡或是boss死亡
        {
            printf("die!\n");
            printf("您的分数是%d分\n", kill);
            break;
        }
        else if (boss.hp<0)
        {
            printf("win!win!win!您击败了比利王，新日暮里将再次重回Deep Dark Fantasy的怀抱！ \n");
            printf("您分数为%d分\n",kill);
            break;
        }
        FlushBatchDraw();           //内存中绘图输出
    }
    EndBatchDraw();      //结束绘图
    mciSendString(" stop ./voice/beijing.mp3 ", NULL, 0, NULL);
    closegraph();
    system("pause");
    return 0;
}
//游戏开始，通过调用其它函数来实现游戏功能
```

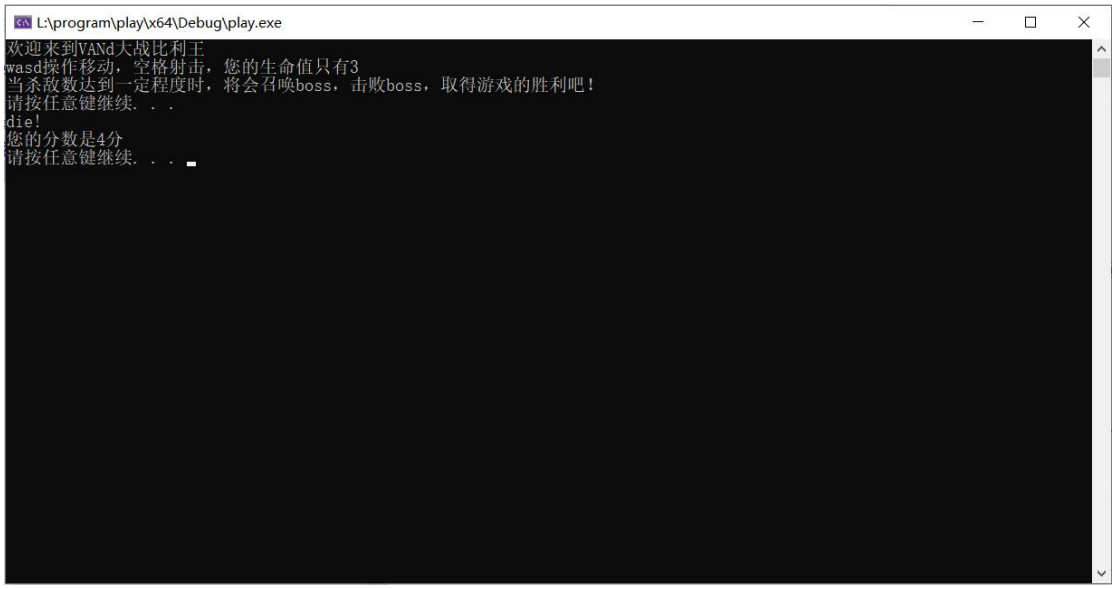
游戏开始前，在循环外打印或调用一些**只使用一遍**的函数如游戏初始化函数。

开始界面展示：



在**循环**中则**反复调用**功能函数来达到动态操作的目的，在循环的最后使用**分支**判断游戏结束条件是否达成，达成则**跳出**循环游戏结束，依据胜负输出不同结果。

失败的结果：



在主函数中使用了五个 easyx 库中的函数：initgraph()生成图形窗口；BeginBatchDraw()开始将内存的内容输出到缓冲区；FlushBatchDraw()将缓冲区输出到屏幕，此函数需要反复调用；EndBatchDraw()结束绘制，closegraph()关闭窗口，回到 cmd 界面。

2）定义数据：

```
int kill = 0;                                   //设置分数，击败敌人即可计分
enum my {
    WIDTH = 691,
    HEIGHT = 431,
    bull_num = 999,
    enemy_num = 25,
    BA,
    MO
};
//定义所需数据如场景长宽，子弹敌人数量，敌人种类

IMAGE bk;                                       //定义背景图
IMAGE img_player[3];                            //定义玩家角色图片
IMAGE img_bull[2];                              //定义玩家子弹图片
IMAGE img_enemy[2][2];                          //定义敌人图片
IMAGE img_boss[2];                              //定义boss图片
IMAGE img_bossbull[2];                          //定义boss子弹图片
int x=0;                                        //用来判断boss移动方向
struct plance{
    double x;
    double y;
    bool live;
    int width;
    int height;
    int hp;
    int type;                                   //定义单位属性，如位置，是否存活，血量，种类
}player,bull[bull_num],enemy[enemy_num],boss,bossbull[bull_num];//定义各种单位
```

定义一些**全局变量**以方便在子函数中进行判断、操作，使用**枚举、结构**定义单位数据，根据需要不同来用单个变量或数组、**二维数组**来储存数据。

3）图片加载函数

```
void loadimg()                                  //加载图片
{

    loadimage(&bk, "./image/bk.png");    //加载背景图

    loadimage(&img_player[0], "./image/van2.png");  //加载自己
    loadimage(&img_player[1], "./image/van.png");
    loadimage(&img_player[2], "./image/van3.png");

    loadimage(&img_bull[0], "./image/go2.png");      //加载子弹
    loadimage(&img_bull[1], "./image/go.png");
    loadimage(&img_bossbull[0], "./image/ebull2.png");
    loadimage(&img_bossbull[1], "./image/ebull.png");

    loadimage(&img_enemy[0][0], "./image/banana2.png");//加载敌人
    loadimage(&img_enemy[0][1], "./image/banana.png");
    loadimage(&img_enemy[1][0], "./image/MO2.png");
    loadimage(&img_enemy[1][1], "./image/MO.png");
    loadimage(&img_boss[0], "./image/bili2.png");
    loadimage(&img_boss[1], "./image/bili.png");
}
//图片加载函数
```

在此函数中使用 **easyx 库**中的 loadimage()函数来将游戏中单位的图片来进行加载，将图片加载到定义的变量当中此处双引号中的路径形式，需要将项目的**字符集改为多字符集**才能正常读取。

4）图像输出函数：

```
100  void gamedraw()                                                    //图片生成
101  {
102
103      putimage(0, 0, &bk);
104      if (player.hp > 1)                                             //玩家高生命时的图片
105      {
106          putimage(player.x, player.y, &img_player[0], NOTSRCERASE);//使用图片位操作达到透明效果
107          putimage(player.x, player.y, &img_player[1], SRCINVERT);
108      }
109      else                                                           //低生命值时变成红色
110      {
111          putimage(player.x, player.y, &img_player[0], NOTSRCERASE);
112          putimage(player.x, player.y, &img_player[2], SRCINVERT);
113      }
114      for (int i = 0; i < bull_num; i++)                             //子弹图片
115      {
116          if (bull[i].live)                                          //判断子弹已生成时打印图片
117          {
118              putimage(bull[i].x, bull[i].y, &img_bull[0], NOTSRCERASE);
119              putimage(bull[i].x, bull[i].y, &img_bull[1], SRCINVERT);
120          }
121      }
122      for (int i = 0; i < bull_num; i++)
123      {
124          if (bossbull[i].live)                                      //判断boss子弹已生成时打印图片
125          {
126              putimage(bossbull[i].x, bossbull[i].y, &img_bossbull[0], NOTSRCERASE);
127              putimage(bossbull[i].x, bossbull[i].y, &img_bossbull[1], SRCINVERT);
128          }
129      }
130       for (int i = 0; i < enemy_num; i++)                           //依据敌人种类打印对应图片
131       {
132           if (enemy[i].live)
133           {
134               if (enemy[i].type == MO)
135               {
136                   putimage(enemy[i].x, enemy[i].y, &img_enemy[1][0], NOTSRCERASE);
137                   putimage(enemy[i].x, enemy[i].y, &img_enemy[1][1], SRCINVERT);
138               }
139               else
140               {
141                   putimage(enemy[i].x, enemy[i].y, &img_enemy[0][0], NOTSRCERASE);
142                   putimage(enemy[i].x, enemy[i].y, &img_enemy[0][1], SRCINVERT);
143               }
144
145           }
146       }
147       if (boss.live)                                                //boss已生成时打印boss图片
148       {
149           putimage(boss.x, boss.y, &img_boss[0], NOTSRCERASE);
150           putimage(boss.x, boss.y, &img_boss[1], SRCINVERT);
151       }
152  }
153   //输出图像函数
```

在此函数前先调用 loadimg()函数进行加载，加载过后才能进行输出。玩家的图片有两种样式，高生命和低生命，以此提醒玩家注意；敌人的图片也有两种形式，以此区分强弱，增强游戏性；子弹、boss 的图片也整合在这个函数中输出。所有图片输出前都会使用 if 结构对其存活状态进行判定，当存活时才进行输出。

输出使用了 easyx 库中的 putimage()函数，此函数接收**四个参数**，需要注意的是，**坐标为 int 形式，但为了游戏性的妥协，不得不将坐标定义为 double 形式**，因为若使用 int 定义

坐标会使得**移动速度过快**，而且图片输出时坐标的精度**对游戏性影响不大**，因此定义为 double 形式。第四个输入的参数，NOTSRCERASE 和 SRCINVERT 是对图片进行**位操作**，将掩码图与原图混合，以此达到消除白边的作用。

单位角色图片展示：

玩家　　一种敌人

本程序中所有加载的图片均由两次 putimage()分别调用掩码图和原图进行**位操作**完成。

5）游戏初始化函数：

```
77  void gameinit()                                          //游戏初始化
78  {
79      loadimg();                                           //图片加载初始化
80      player.x = 0;
81      player.y = HEIGHT / 2;
82      player.live = true;
83      player.hp = 3;                                       //玩家初始化
84      for (int i = 0; i < bull_num; i++)                   //子弹初始化
85      {
86          bull[i].x = 0;
87          bull[i].y = 0;
88          bull[i].live = false;
89      }
90      for (int i = 0; i < enemy_num; i++)
91      {
92          enemy[i].live = false;
93          enemyhp(i);
94      }
95      mciSendString(" play ./voice/beijing.mp3 ", NULL, 0, NULL);//背景音乐初始化
96      mciSendString(" play ./voice/beijing.mp3 repeat ", NULL, 0, NULL);
97      boss.hp = 60;                                                //boss生命值初始化
98  }
99  //游戏初始化函数
```

此函数中先调用加载函数，因为只需要**调用一次**。然后初始化玩家位置，将玩家的位置放在窗口左边的中间，将存活状态设置为存活，血量设置为 3。

初始化子弹，用 for **循环**将子弹数组填满，位置初始化在窗口左上角，但此时状态设置为了 false，因此不会出现。敌人同理，但先将存活状态设置为 false，其余操作在另一函数中完成，其作用是生成敌人并分化。Boss 的血量也**提前在此初始化**，防止出现问题。

然后设置背景音乐循环播放。此处调用 mciSendString()函数，此函数是包含在 mmsystem.h **和** Winmm.h 中的一个**多媒体播放函数**，此函数接收 4 个参数，但我们只需要输入音频文件的路径，其余的**设置为 0** 即可。

6）接下来就是刚才调用的敌人分化函数：

```
58  void enemyhp(int i)                                          //决定生成敌人种类
59  {
60          if (rand() % 10 == 0||rand()%10==1)                  //强力敌人生成属性
61          {
62              enemy[i].type = MO;
63              enemy[i].hp = 5;
64              enemy[i].width = 65;
65              enemy[i].height = 40;
66          }
67          else                                                 //普通敌人生成属性
68          {
69              enemy[i].type = BA;
70              enemy[i].hp = 2;
71              enemy[i].width = 34;
72              enemy[i].height = 35;
73          }
74
75  }
76   //敌人分化函数
```

此函数是为了增加游戏性。

简单使用一个 **if 分支**，判断条件为**随机数**，强大一些的敌人生成概率低。并设置好敌人的长度高度，在接下来的函数中需要调用。

7）敌人生成函数：

```
168  void create_enemy()
169  {
170      for (int i = 0; i < enemy_num; i++)
171      {
172          if (!enemy[i].live&&!boss.live)              //当敌人被消灭后且boss未出现时敌人重生
173          {
174              enemy[i].live = true;
175              enemy[i].x = WIDTH-35;
176              enemy[i].y = rand()%(HEIGHT-40);         //依据随机数来随机安排重生位置
177
178              break;
179          }
180      }
181      if (kill == 100||kill==101)                      //当分数达标时生成boss
182      {
183
184          mciSendString(" play ./voice/here.mp3  ", NULL, 0, NULL);
185          boss.live = true;
186          boss.width = 104;
187          boss.height = 100;
188          boss.x = WIDTH-100;
189          boss.y = HEIGHT-boss.height;
190      }
191
192  }
193   //敌人生成函数
```

第一个 **for 循环**中，将敌人**数组遍历**，全部设置为 true，并将横坐标设置在右端，纵坐标随机生成。**if** 判断是为了在 boss 登场以后小怪不再刷新，降低游戏难度的同时**降低内存消耗和计算占用，避免卡顿**。第二个 **if** 是判断 boss 出现条件，当杀敌数超过一百后登场，此处的 100 或 101 是因为敌人击杀后分数为 **1 或 2**，防止被跳过或多次执行，因为此函数在主函数的循环中，此部分只需调用一次，将 boss 生成在右下角并播放登场音效。

敌人移动函数：

```
194  void enemymove(double speed)
195  {
196      for (int i = 0; i < enemy_num; i++)
197      {
198          if (enemy[i].live)                    //当敌人存活时控制敌人移动
199          {
200              enemy[i].x -= speed;
201              if (enemy[i].x < 0)               //敌人出版边时移除
202                  enemy[i].live = false;
203          }
204      }
205  }
206  //敌人移动控制函数
```

很简单的一个循环，当敌人存活时不断向左边移动，当敌人越过左版边后清除，**减少内存占用。**

8）玩家移动函数：

```
221  void playermove(float speed)
222  {   //使用windows非阻塞函数检测键盘输入wasd
223      //传入一个参数控制移动速度
224      if (GetAsyncKeyState('W'))
225          if(player.y≥-20)
226          player.y -= speed;
227
228      if (GetAsyncKeyState('S') )
229          if(player.y≤HEIGHT-20)
230          player.y += speed;
231
232      if (GetAsyncKeyState('A'))
233          if(player.x≥-20)
234          player.x -= speed;
235
236      if (GetAsyncKeyState('D') )
237          if(player.x≤WIDTH-30)
238          player.x += speed;
239      static DWORD t1, t2;//控制射击间隔
240      if (GetAsyncKeyState(VK_SPACE) && t2 - t1 > 200)//敲击空格且经过一定时间间隔后才能成功发射
241      {
242          create_bull();
243          mciSendString(" play ./voice/woo.mp3 ", NULL, 0, NULL);//发射音
244          t1 = t2;
245      }
246      t2 = GetTickCount();//返回时间间隔
```

有另一种控制移动的方法，即使用 getch()和 switch 来进行判断输入，但此种方法会**阻塞**，且**不能连续输入，显著降低游戏的响应速度**故使用了包含在 windows.h 中的 GetAsyncKeyState()函数，可以**流畅的进行判定，避免卡顿**。此外，将射击判定也写入其中，**局部定义 DWORD 类型**记录时间，当敲击空格且时间间隔大于 200ms 时才能射击，调用一次子弹创建函数并在射击时调用函数 mcisendstring()来增加音效。

9）玩家子弹生成函数：

```
154  void create_bull()
155  {
156      for (int i = 0; i < bull_num; i++)
157      {
158          if (!bull[i].live)                          //将子弹位置生成在玩家面前
159          {
160              bull[i].x = player.x+10;
161              bull[i].y = player.y;
162              bull[i].live = true;
163              break;                                  //一次生成一发后跳出
164          }
165      }
166  }
167  //生成玩家子弹函数
```

依然使用 for **循环**遍历数组直到找到当前射出的子弹，将子弹位置移动到玩家前面并激活后跳出。

10）子弹移动控制函数：

```
207  void bullmove()
208  {
209      for (int i = 0; i < bull_num; i++)
210      {
211          if (bull[i].live)                           //生成子弹后控制子弹移动
212          {
213              bull[i].x += 0.1;
214              if (bull[i].x > WIDTH + 10)
215                  bull[i].live = false;               //出版边后移除
216
217          }
218      }
219  }
220  //玩家子弹移动控制函数
```

for 循环是为了找到当前子弹，找到后如果状态为激活，则向左移动，并在出版边后清除**减少内存占用**。

11）受击判定函数：

```
262  void hit()
263  {
264      for (int i = 0; i < enemy_num; i++)
265      {
266          if (!enemy[i].live)//敌人是否存活受击判定
267              continue;
268          for (int k = 0; k < 999; k++)
269          {
270              if (!bull[k].live)//子弹是否存在受击判定
271                  continue;
272              if (bull[k].y+20 > enemy[i].y && bull[k].y<enemy[i].y + enemy[i].height
273                  && bull[k].x≥enemy[i].x && bull[k].x < enemy[i].x + enemy[i].width)//子弹命中目标是开始判定伤害
274              {
275                  bull[k].live = false;                                   //命中后移除子弹
276                  enemy[i].hp--;                                          //受击伤害
277                  if(enemy[i].type==MO)                                   //依据敌人种类播放不同敌人的受击音效
278                  mciSendString(" play ./voice/enemyhit.mp3 ", NULL, 0, NULL);
279                  else
280                  mciSendString(" play ./voice/enemyhit1.mp3 ", NULL, 0, NULL);
281              }
282
283          }
```

```
284             if (player.y + 40 > enemy[i].y && player.y < enemy[i].y + enemy[i].height    //撞击判定
285                 && player.x ≥ enemy[i].x && player.x < enemy[i].x + enemy[i].width)
286             {
287                 static DWORD t1, t2;        //玩家受击后给予无敌时间
288                 if (t2 - t1 > 500)
289                 {
290                     player.hp--;
291                     mciSendString(" play ./voice/playerhit.mp3 ", NULL, 0, NULL);//玩家受击音效
292                     t1 = t2;
293                 }
294                 t2 = clock();
295 
296                 if (player.hp ≤ 0)              //玩家生命值判断是否死亡
297                 {
298                     player.live = false;
299 
300                 }
301             }
302             if (enemy[i].hp ≤ 0)                //敌人生命值判断是否死亡
303             {
304                 enemy[i].live = false;
305                 if (enemy[i].type == MO)
306                 {
307                     enemy[i].hp = 3;
308                     kill += 2;                  //击杀后奖励分数
309                 }
310                 else
311                 {
312                     enemy[i].hp = 1;
313                     kill += 1;
314                 }
315             }
316         }
317     }
```

首先判定敌人和子弹存活，存活时才能继续，若不存活则直接**略过本次循环**直接寻找下一个单位，**减少计算量。**当子弹命中敌人时（坐标位置达到图片的重叠范围时），将子弹移除，并削减敌人的生命值，播放敌人受击音效。敌人与玩家碰撞时同理，不同的是会给玩家 **500ms 的无敌时间**（使用与子弹射击间隔相同的判定方法），防止碰到敌人瞬间被秒杀。然后判定敌我血量，当血量减至 0 时判定死亡。若玩家死亡，此时满足游戏结束条件，游戏结束；敌人死亡，根据种类不同加分不同，强力敌人加两分普通敌人加一分。判定敌人死亡时，将敌人血量重置，准备再次刷新。

12）计时器函数：

```
250 int timer(unsigned int s)
251 {
252     static DWORD t1 = 0, t2 = 0;
253     if (t2 - t1 > s)
254     {
255         return 1;
256         t1 = t2;
257     }
258     t2 = clock();
259     return 0;
260 }
```

程序中所有的计时代码段都与此类似，均可替换，但此函数只在主函数中调用，用来控制敌人刷新速度。因此，本程序还可以改进，将全部计时用此函数操作。

13）BOSS 系列函数：

因为单独引入了一个 boss，不便再将其整合进入小怪的控制函数中（减少 bug），因此重新为 boss 编写了控制函数。

```
void bosshit()
{
    for (int k = 0; k < 999; k++)
    {
        if (!bull[k].live)//同上敌人受击判定
            continue;
        if (bull[k].y + 20 > boss.y && bull[k].y < boss.y + boss.height
            && bull[k].x ≥ boss.x && bull[k].x < boss.x + boss.width)
        {
            bull[k].live = false;
            boss.hp--;
            mciSendString(" play ./voice/bosshit.mp3 ", NULL, 0, NULL);//boss受击音效
        }

    }
    for (int k = 0; k < 999; k++)                    //boss子弹对玩家判定伤害
    {
        if (!bossbull[k].live)
            continue;
        if (bossbull[k].y + 20 > player.y && bossbull[k].y < player.y + 40
            && bossbull[k].x ≤ player.x + 25 && bossbull[k].x ≥ player.x)
        {

            bossbull[k].live = false;
            player.hp--;
            mciSendString(" play ./voice/playerhit.mp3 ", NULL, 0, NULL);
            break;
        }
    }

    if (player.y + 40 > boss.y && player.y < boss.y + boss.height
        && player.x ≥ boss.x && player.x < boss.x + boss.width)//boss对玩家碰撞伤害判断并给予无敌时间
    {
        static DWORD t1, t2;
        if (t2 - t1 > 500)
        {
            player.hp--;
            mciSendString(" play ./voice/playerhit.mp3 ", NULL, 0, NULL);
            t1 = t2;
        }
        t2 = clock();
    }
        if (player.hp ≤ 0)
        {
            player.live = false;

        }
        if (boss.hp ≤ 0)
            boss.live = false;          //双方死亡判断

}
//玩家boss受击判断函数
```

与小怪的受击判定基本相同，只是修改了部分内容，如变量名、声音。

```
371  void boss_attack()
372  {   if(boss.live)
373      for (int i = 0; i < bull_num; i++)//判断boss存活，若是，则自动开始攻击，类似玩家射击
374      {
375          if (!bossbull[i].live)
376          {
377              bossbull[i].x = boss.x;
378              bossbull[i].y = boss.y + boss.height / 2;
379              bossbull[i].live = true;
380              break;
381          }
382      }
383  }
384   //boss攻击函数
```

此函数控制 boss 的攻击，这是 boss 与小怪最大的区别，boss 会和玩家一样发射子弹，代码和玩家子弹生成函数基本一致。

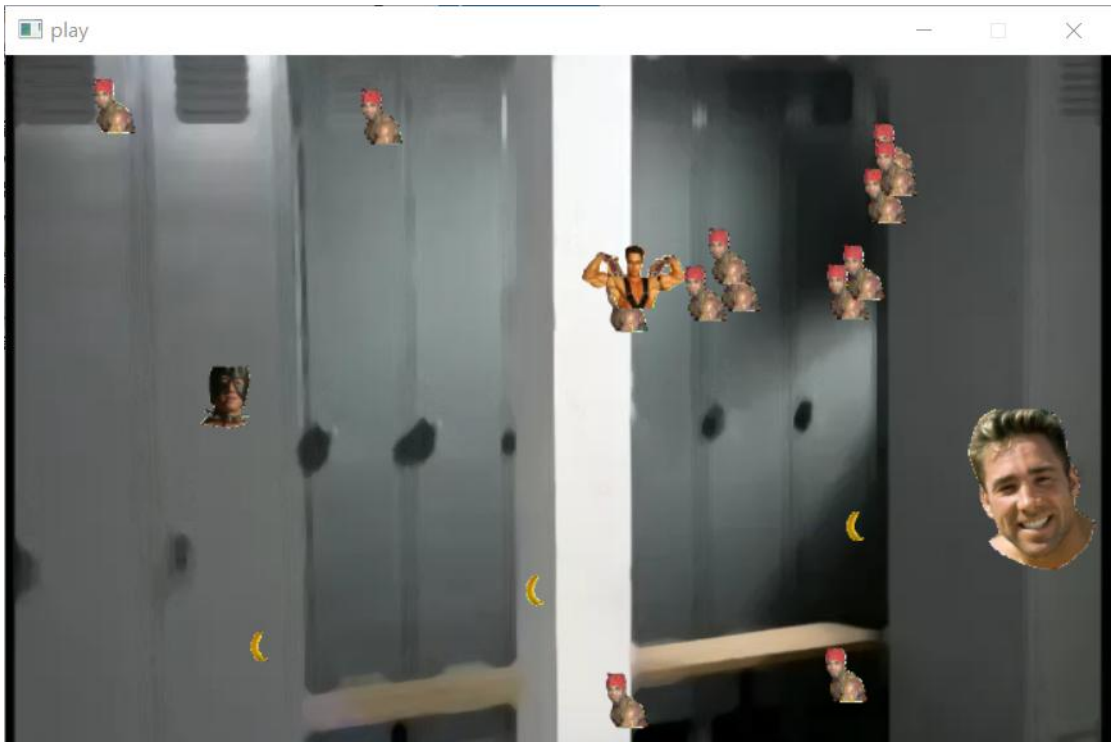
```
385  void bossmove(double speed)
386  {   //传入boss移动速度参数
387      if (boss.live)
388
389      {
390          if (x % 2 == 0)//开始时boss生成在下，开始向上移动，每碰撞一次版边转向，偶次向上移动至顶部，奇次向下移动至底部
391          {
392              boss.y -= speed;
393
394              if (boss.y - speed < 0)
395              {
396                  x++;
397
398              }
399          }
400          if (x % 2 != 0)
401          {
402              boss.y += speed;
403              if (boss.y + speed > HEIGHT - boss.height)
404                  x++;
405          }
406
407
408
409          static DWORD t1, t2;//boss射击间隔
410          if (t2 - t1 > 1000)
411          {
412              boss_attack();
413              mciSendString(" play ./voice/bossattack.mp3 ", NULL, 0, NULL);//boss射击音效
414              t1 = t2;
415          }
416          t2 = clock();//返回时间间隔
417      }
418  }
```

Boss 移动控制函数，boss 会在右端不断上下移动，但我没有更好的方法去控制 boss 改变朝向，因此开始定义的变量 x 在此处使用，因为 boss 初始在下，x=0，便设置 x 为偶数时向上，否则向下，改变条件是 boss 碰到版边，碰到一次 x+1。此外，boss 攻击也整合入此函数，类似玩家移动控制函数，除了 boss 是自动射击，且射击速度慢一些。Boss 在设计时也会像玩家一样播放专属射击音效。

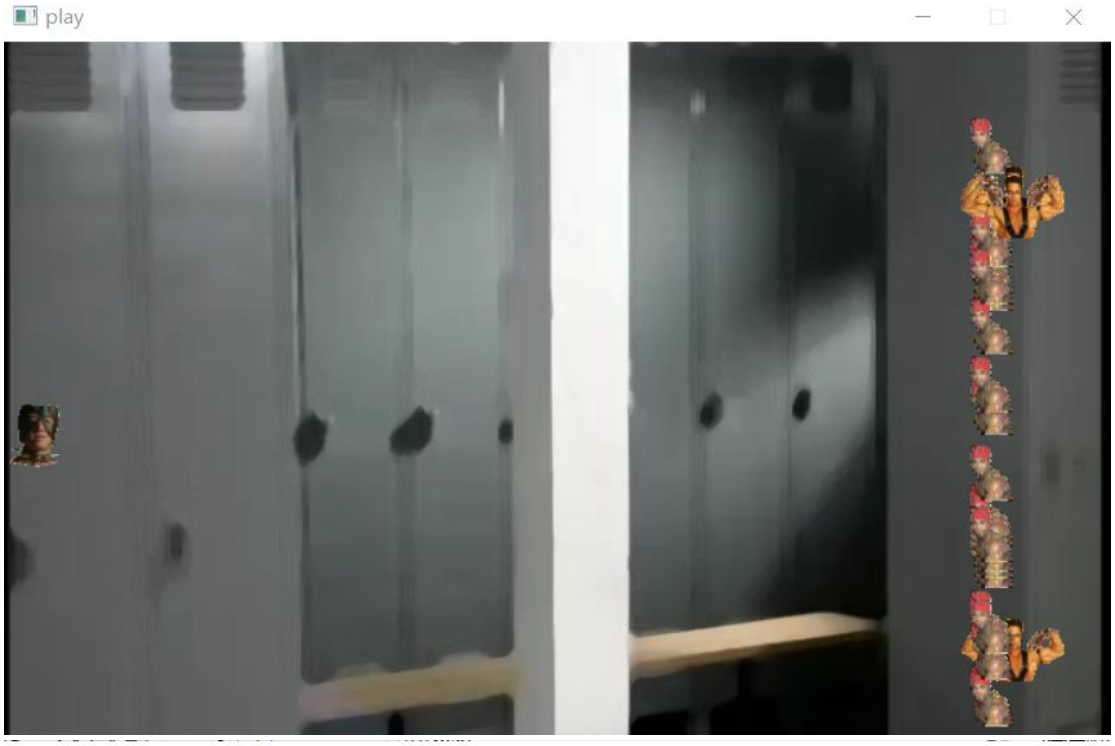
```
420  void bossbullmove()
421  {
422      for (int i = 0; i < bull_num; i++)//boss子弹移动控制，同玩家子弹移动
423      {
424          if (bossbull[i].live)
425          {
426              bossbull[i].x -= 0.1;
427              if (bossbull[i].x < 0)
428                  bossbull[i].live = false;
429
430          }
431
432      }
433
434  }
435  //boss子弹移动控制函数
```

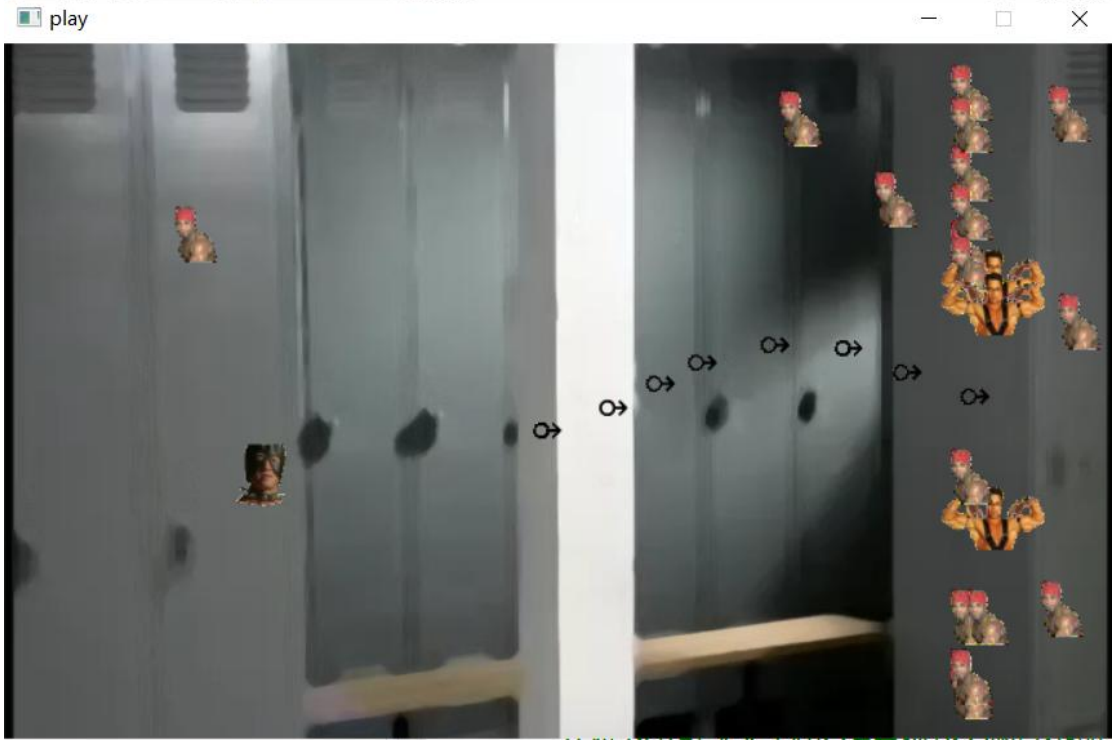
Boss 子弹移动控制，和玩家的完全相同，只是方向相反。

Boss 出现后在移动射击，且小怪不再刷新：



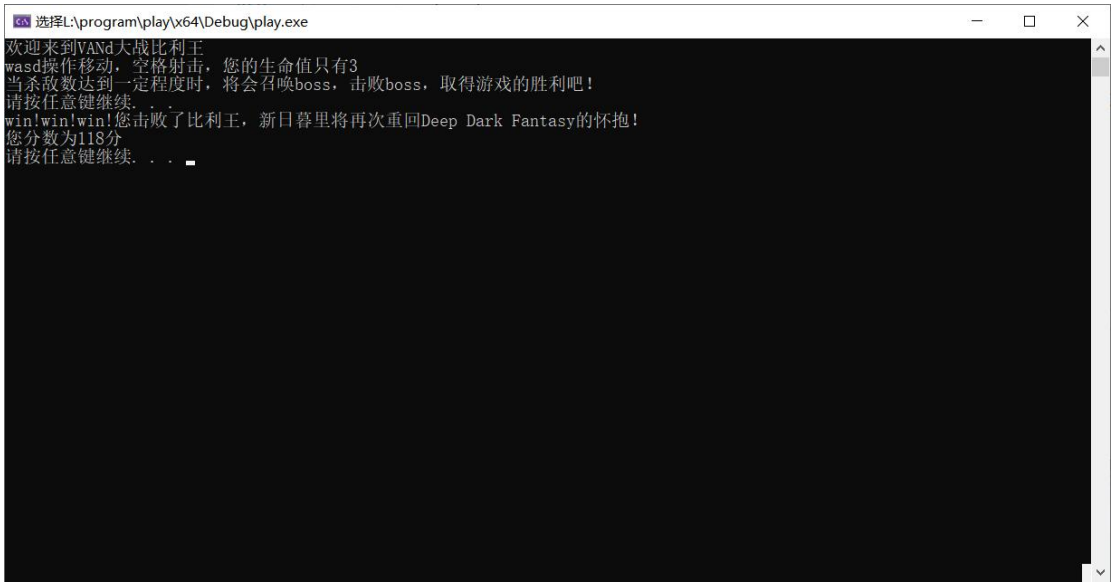
三、游戏效果展示：





玩家在射击；

玩家低血量时的特效；



游戏胜利；

四、实验过程中遇到的问题及解决方案：

1.图片加载有白色边框。通过查找资料后找到了位操作方法，但试用后出现了图片反色等问题，后反复实践操作解决了这个问题。

2.boss 的引入。原本是想将 boss 的相关控制整合进入小怪控制，但出现了很多问题。如不便控制出现时机。原本使用 Switch 分支进行选择生成 boss 或小怪，但过于麻烦，通过独立定义 boss 变量名而不是整合敌人数组解决。解决后又出现了 boss 无敌的问题，通过独

立编写判定函数解决。Boss 判定正常后又出现了无法移动并且无敌问题，原来是 boss 生成的判断条件为分数大于 100，使得 boss 不断被重置，导致了问题的产生。通过将生命值初始化分离出去，改变生成判断条件并防止条件被略过，多次调试修改后解决了问题。

在编写 boss 攻击判定时，出现了判定异常，只有在玩家攻击时才能判定伤害。反复测试后发现是玩家移动控制函数与 boss 攻击判定出现了冲突，只有在按下空格时才能判定，独立编写 boss 判定后解决。

解决判定异常后又出现了判定速度过快，导致接触到 boss 子弹时被瞬间秒杀，原因是没有像碰撞一样设置无敌时间，但单独编写无敌时间又很麻烦，解决办法是在 for 循环 if 判定最后加入 break 跳出，只要判定一次后立刻跳出，防止反复判定。

3. 函数接收路径问题。因为熟练程度不够，路径输入格式不太了解，只能以程序中所写的那样输入，但输入后会编译错误，上网查阅相关信息后将字符集设置改为多字符集后解决。

4. 玩家移动控制函数原本使用 switch 控制，用 conio 库中的 getch()获取输入，但此种方法卡顿严重。

5. 游戏结束后在非编译器中运行闪退，加入 system(“pause”)后反而会卡住，反复研究后发现是因为图形窗口未关闭，加入 closegraph 函数后解决。

6. Boss 登场音效有时会发生异常，播放时机不确定，暂未发现原因，无法解决。

五、分析改进空间

1）已经编写了计时器函数，可以将全部需要计时的地方用此函数控制而不是再定义；

2）函数的调用可已进一步整合，例如将 boss 系列的函数嵌套起来调用，在主函数中更加清晰简洁；

3）功能函数可以整合，但限于编写过程中反复修改 bug 的经历，时间受限的条件下个人较难实现；

4）数据结构可以进一步优化，如在外部定义的一些变量，其实并没有必要在外部定义；

5）开始结束时也可以使用图形化界面，限于精力没有进一步打磨；

6）游戏性还有无限的可能，如奖励玩家升级，增加关卡设计，增加小怪射击，射击形式多样化，优化 boss 行动逻辑……

7）指针使用问题。程序力求化繁为简，复杂的问题能用简单的方式解决就用简单的方式解决，硬凑指针使用不仅思路不会更清晰，还可能出现更多 bug，故除了特定函数需要外没有使用专门的指针。

六、详细说明

wasd 操控移动，大小写中英文均可识别，空格射击，进入后记得将输入法中文关闭，虽然不影响输入但会生成输入框遮挡。

玩家生命值为 3，受击后会有音效，生命值为 1 时会变红。

第一波敌人生成时，强力敌人生命值为 3，普通敌人生命值为 2（防止分数开始时增长过快），刷新以后普通敌人的生命值会降低为 1。击败强力敌人加 3 分，普通敌人加 1 分。每一波敌人最多为 25 个。

当分数到 100 时，boss 出现，为了降低难度此时小怪不再刷新，专心与 boss 作战。Boss 血量为 60，初始生成位置右下角，只会上下移动射击，boss 射击子弹和自身移动速度较快，注意躲避。

玩家可以自行对所有上述红色数据进行更改以修改难度。

七、心得体会

个人的力量终究是有极限的，对于某些创意想法难以实现，但好处是思路的连贯，对于代码极度熟悉方便找出 bug，能够做出自己真正想做的东西。

做这个项目时，一开始是有一个大致的模板，但里面细节得的东西，是需要自己去填充的。如 boss 的加入，完全是个人设计出来的，每写一点就调试调试，一调试就出 bug，又反复修改。有灵感的时候马上就能想到 bug 出在哪里，立刻就能设计出解决思路；没有灵感的时候怎么都不知道错在哪里，好不容易找到了又不知道怎么改，改一下看一下，弄好了就行。

亲自敲下代码感受一个项目从无到有的感觉很是奇妙，令人欣喜。

作为本人的第一个大项目，做的还是很粗糙，后续还会对这个程序反复打磨，设计更多的功能，希望能够把它真正的做成一个有趣的游戏。