北京理工大學

汇编接口上机实验报告

北京理工大学汇编接口小组项目实验报告

Compiling the experimental report of interface group project of Beijing Institute of Technology

学　　院：计算机学院

专　　业：计算机科学与技术

学生姓名：杨汶锦、王岩、蒙思洁、吴一凡

学　　号：1120193624、27、02、32

指导教师：张全新

2022 年 6 月 13 日

目 录

第 1 章 游戏介绍

本项目制作的是一款射击类游戏。本游戏支持双人对战，两名玩家可以通过控制键盘上的“←”“→”两个按钮来完成射击。射出的子弹在飞行路径上可以触碰障碍物来完成属性的更新，从而获得增益和减益的不同效果。人物中弹会扣掉血量，当一方血量被扣完，则该玩家失败，另一位玩家胜利。

1.1 小组成员

- 姓名：杨汶锦 | 学号：1120193624 | 班级：07111908
- 姓名：王岩 | 学号：1120193627 | 班级：07111908
- 姓名：吴一凡 | 学号：1120193632 | 班级：07111908
- 姓名：蒙思洁 | 学号：1120193602 | 班级：07111908

第 2 章 开发环境

代码的编写主要是在 Visual Studio 中完成的。

Visual Studio 版本为：Microsoft Visual Studio Community 2019。

Visual Studio 中使用的 MSVC 工具集版本：MSVC-v14.24。

汇编开发工具包文件：masm32。

贴图的处理使用 Adobe Photoshop，版本: 20.0.6 20190724.r.80 2019/07/24: 1207344 x64

第 3 章 设计思路及实现过程

3.1 程序整体框架

整个程序利用了 acllib 库的 API 来完成，acllib 其底层还是用到 windows.inc 的许多东西，但使用起来更加顺手简洁。

程序的运行结构和利用 windows.inc 的那种消息派发解释循环的结构类似，也有其固定格式：在 init_first，init_second 间画好主界面，先设置初始化窗口 initWindow：

```
main proc c
    invoke init_first  ;初始化绘图环境
    invoke initWindow, offset winTitle, 250, 30, 800, 600 ;左上角的坐标, 窗体的宽高
    ...
    invoke start_menu;画初始菜单以及初始值
    invoke init_second
main ENDP
END main
```

注册三个类，分别有键盘、鼠标、和计时器类用于触发事件（在之后具体解释）。

画游戏的主菜单页面是在子程序里。

计时器类依据不同的标志变量，来调用不同的函数，程序调用 draw_game 不断更新。

鼠标类用来触发点击事件：在鼠标点击后获得点击坐标，判断位置如果在 start 处则开始游戏，调用游戏的初始化设置，以及调用触发计时器，以 30ms 为间隔来不断触发更新。

人物结构，子弹结构，道具结构的初始化，有关位置、大小、方向、速度的问题。而对于道具的初始化，则要用到一个随机函数来完成：

```
getRand proc c uses ecx edx rand_num: dword
    ;设置随机种子
    push 0
    call crt_time
    add esp,4

    add eax,seed
    mov seed,eax;seed更新

    push eax
```

```
        call crt_srand
        add esp,4

        invoke crt_rand
        mov edx, 0
        mov ecx, rand_num
        div ecx
        mov eax, edx;返回余数
        ret
    getRand endp
```

getRand 中的原理是：利用时间做种子，来生成随机数。但由于汇编中执行速度非常快，在最初的时候，由于时间种子一样，得到的位置是一致的。最后解决方法为不用时间做种子，自己定义了一个 seed，在每次调用 getRand 时做更新，因此得到了每次不同的值。

键盘类用来触发人物的有子弹时发弹、无子弹时改变方向的事件。

3.2 设计思路和实现过程

在设计的过程中，我们对程序的功能进行了分解。主要将其按照页面和逻辑划分。分为主菜单页面、正式游戏页面、游戏结束页面。在主菜单页面中，主要需要实现页面跳转逻辑、图片插入逻辑，需要维护一个按钮结构体；在正式游戏页面里，主要需要实现图片插入逻辑、图片切换逻辑、人物和子弹移动逻辑、道具功能逻辑、子弹射中判定逻辑、事件响应逻辑，需要维护人物属性结构体、子弹属性结构体、道具属性结构体。主菜单页面和正式游戏页面的衔接过程由页面跳转逻辑实现。正式游戏页面和游戏结束页面的跳转过程由子弹射中判定逻辑后对游戏人物生命值的调整来实现。

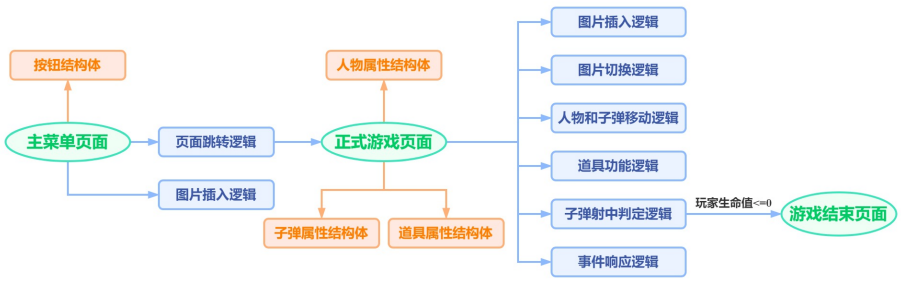

图 3-1 游戏逻辑设计分解图

3.2.1 页面跳转逻辑

页面跳转逻辑主要控制游戏中的各种界面的衔接。游戏主要有两个页面，第一个是主菜单页面，第二个是正式游戏页面。主菜单页面作为打开游戏后，玩家能看到的第一个界面，拥有的最主要的功能就是作为一个首界面，实现向正式游戏页面跳转的逻辑。在主菜单页面中，设置两个按钮，一个为“Start”，一个为“Exit”。“Start”按钮在点击后的功能是跳转去“正式游戏页面”。“Exit”按钮在点击后的功能是退出程序运行。

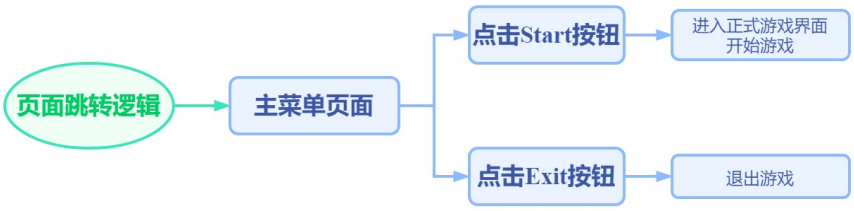

图 3-2 页面跳转逻辑示意图

当前界面需要维护的是一个结构体类型 MyButton，它的任务是记录当前结构体变量的图片范围——左边界 left、右边界 right、上边界 top、下边界 bottom。在具体的代码实现中，定义为 start_button 和 exit_button 两个结构体变量。

```
MyButton struct
    top dd  ?
    left  dd  ?
    right dd  ?
    bottom  dd  ?
MyButton ends
```

页面跳转部分主要依靠全局变量 curWindow 来控制。在当前页面为主菜单页面时，为 curWindow 赋值为 0；如果当前页面为正式游戏页面，则为 curWindow 赋值为 1。

3.2.2 图片插入逻辑和图片切换逻辑

图片插入逻辑主要控制的是游戏中的界面美观程度和游戏的视觉体验感。在对图片进行制作和处理后，我们确定了所有要插入图片的位置和图片素材，绘制了每个页面的各个图片素材应该出现在的位置的示意草图。

我们把图片分为五类，分为按钮、背景、子弹、人物、道具。

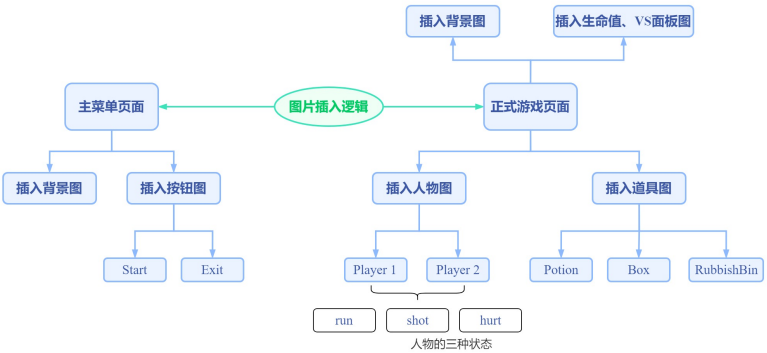

图 3-3 图片插入逻辑示意图

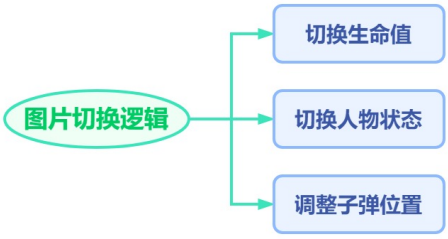

图 3-4 图片切换逻辑示意图

主菜单页面只关注主菜单背景图和“Start”、“Exit”按钮图的绘制，因此在start_menu 子程序中，对主菜单页面进行绘制即可，并且在这里加入判断标识curWindow=0，用来表示这是主菜单页面。

```
start_menu proc C
    ;载入图片
    invoke loadImage, offset page1_Bg, offset imgBg
    invoke loadImage, offset page1_Title, offset imgTitle
    invoke loadImage, offset page1_Start, offset imgStart
    invoke loadImage, offset page1_Exit, offset imgExit

    ;显示主界面
    invoke beginPaint
    invoke putImageScale, offset imgBg, 0, 0, 800, 600
    invoke putImageScale, offset imgTitle, 200, 100, 400, 100
    invoke putImageScale, offset imgStart, 260, 300, 240, 100
    invoke putImageScale, offset imgExit, 260, 450, 240, 100
    invoke endPaint

    ;设置参数:
    mov curWindow,0
    ret
start_menu endp
```

接下来，我们调用自己写的 draw_game 子程序来实现正式游戏页面的图片绘制、整体图片根据帧数更新位置的功能。正式游戏页面中需要使用到背景、人物、子弹、道具。

在本部分代码的设计过程中，首先我们明确，人物、子弹是同时具有多种属性的。因此我们编写了人物和子弹的结构体，并分别命名为“person”“Bullet”。在整个程序完成编写后，这两个结构体拥有的属性分别如下：

```
person struct
    life  dd  ?;生命值
    pos_x dd ?;横坐标--不变定死
    pos_y dd ?;纵坐标--上下移动
    size_x  dd  ?;大小
    size_y  dd  ?
    dir dd  ?;移动方向--1为上-1为下
    speed dd  ?;速度大小--
    bullet  dd  ?;子弹数
    is_hit dd ?;是否击中或越界
```

```
    person ends
    person1 person<>
    person2 person<>

    Bullet struct
        show  dd  ?;显示与否
        pos_x dd ?;横坐标
        pos_y dd ?;纵坐标
        dir_x dd  ?;移动方向--1为右，-1为左
        dir_y dd  ?;纵向移动方向 ---用于反弹
        size_x  dd  ?;大小
        size_y  dd  ?
        speed_x dd  ?;速度大小
        speed_y dd  ?
    Bullet ends
    bullet1 Bullet<>
    bullet2 Bullet<>
```

在画背景时，主要使用 draw_bg 子程序。该子程序绘制背景、VS 图、人物生命值图，并且通过对 person 结构体中，玩家控制人物的血量进行判断，来决定绘制几个生命值图像（气球）。

在 draw_bg 中我们使用 ACLLIB 提供的库函数 loadImage 加载图片源；然后使用库函数 putImageScale 将图片以指定大小展示在前端的指定位置上。而后我们需要根据 person 结构体的 life 变量来判定生命值，从而使用条件判断结构在指定位置画出生命值。具体的代码实现如下：

```
    draw_bg proc
        invoke loadImage, offset page2_Bg, offset imgBg2
        invoke loadImage, offset page2_vs, offset imgVS
        ...
        ;分数显示判断
        .if person1.life>=1
            invoke putImageScale, offset imgCharLife1, 145, 495, 30, 50
            .if person1.life>=2
                invoke putImageScale, offset imgCharLife1, 190, 495, 30, 50
                .if person1.life>=3
                    invoke putImageScale, offset imgCharLife1, 235, 495, 30, 50
                    .if person1.life>=4
                        invoke putImageScale, offset imgCharLife1, 280, 495, 30,
    50
```

```
                        .if person1.life==5
                            invoke putImageScale, offset imgCharLife1, 325, 495,
    30, 50
                        .endif
                    .endif
                .endif
            .endif
        ;.elseif person1.life<=0
        ; invoke game_over
        .endif
        ...
        ret
    draw_bg endp
```

在画人物时，首先使用 loadImage 加载人物的不同状态的图片，然后通过判断 person 结构体的 bullet 和 is_hit 标志变量来判断人物的形态——如果子弹数为 0，则人物当前应该属于“run”的普通行走状态；如果子弹数为 1，则人物当前应该属于“shot”的举枪状态。不管人物当前有几颗子弹，只要它中弹，就切换它的贴图为“die”图片。这里的 is_hit 的值表示其中弹图片显示的帧数。具体的条件判断实现代码如下：

```
    draw_man proc
        invoke loadImage, offset page2_char1_run, offset imgCharRun1
        invoke loadImage, offset page2_char2_run, offset imgCharRun2
        ...
        ;通过角色拥有子弹数判断人物形态
        .if person1.bullet==1
            .if person1.is_hit>0;判断中弹
                invoke putImageScale, offset imgCharDie1, person1.pos_x,
    person1.pos_y,  person1.size_x,  person1.size_y
                sub person1.is_hit,1
            .endif
            .if person1.is_hit==0
                invoke putImageScale, offset imgCharShot1, person1.pos_x,
    person1.pos_y, person1.size_x,  person1.size_y
            .endif
        .endif
        ...
        ret
    draw_man endp
```

在画子弹时，要考虑子弹是在人物举枪以后才能被发射出来，两位玩家操控 Player1 和 Player2，分别使用键盘上的“←”和“→”来控制 Player1 和 Player2 的发弹过程，称这两个按键是“发弹键”。子弹的原图是一个直径较小的纯色圆。在人物按下自己对应的发弹键后，人物贴图转换，未触碰道具时，子弹将水平向远离发弹人的方向匀速移动；如果触碰道具，将根据道具的功能来完成子弹的方向调整以及伤害调整。对于子弹结构体中的 show 属性，当 show 的值为 1 时需要画子弹；show 的值为 0 时不需要画子弹。

```
draw_bullet proc p1:dword,p2:dword
    invoke printf,offset coord,p1,p2

    .if p1==1;show为0则不画
        invoke loadImage, offset page2_char1_bul, offset imgCharBul1
        invoke putImageScale, offset imgCharBul1,bullet1.pos_x,
bullet1.pos_y, bullet1.size_x, bullet1.size_y
    .endif

    .if p2==1
        invoke loadImage, offset page2_char2_bul, offset imgCharBul2
        invoke putImageScale, offset imgCharBul2,bullet2.pos_x,
bullet2.pos_y, bullet2.size_x, bullet2.size_y
    .endif
    ret
draw_bullet endp
```

在画道具时，在图片插入逻辑中，关键点在于，要实现道具的随机出现效果。此效果的实现代码放置在 draw_prop 子程序、getRand 子程序、game_init 子程序、道具属性结构体中。道具属性的结构体主要的目标是记录道具的横纵坐标、大小尺寸。道具分三种，分别拥有变向、增强、降速这三种功能，因此初始化为三类结构体变量 Prop_Bounce、Prop_Slow、Prop_Big。

```
Prop struct
    pos_x dd ?;横坐标
    pos_y dd ?;纵坐标
    size_x  dd  ?;大小
    size_y  dd  ?
    ;is_hit dd  ?;是否击中
Prop ends
```

draw_prop 子程序主要是加载绘制道具图片。getRand 子程序用来“随机取数”，在 getRand 子程序被调用时，参数 rand_num 由我们根据想要的个数和图片素材的坐

标指定给出，然后随机一个在此范围内的数。game_init 子程序用来“按照随机数放置道具”。在 game_init 子程序中，我们依靠时间初始化来随机一个 seed，然后调用 getRand 子程序以达到随机取数的效果，且随机数的范围要小于 rand_num。

```
draw_prop proc
  invoke loadImage, offset page2_box, offset imgObBox
  invoke loadImage, offset page2_rub, offset imgObRubBin
  invoke loadImage, offset page2_potion, offset imgObPotion

  mov ebx,0
  mov edi,0
  .while ebx<Bounce_Num
    invoke putImageScale, offset imgObRubBin,(Prop ptr Prop_Bounce[edi]).pos_x,(
   Prop ptr Prop_Bounce[edi]).pos_y, (Prop ptr Prop_Bounce[edi]).size_x,(Prop
   ptr Prop_Bounce[edi]).size_y
    add edi,type Prop ;Prop大小
    inc ebx
  .endw
    ...
  ret
draw_prop endp
```

3.2.3 人物和子弹移动逻辑

子弹和人物的移动实现是通过定时器不断重复调用构成的循环来实现的。每一次循环都被看做一个帧，我们在循环体内更改子弹元素和人物元素的坐标位置并在每个循环中重新绘制，利用人眼的视觉暂留效果，即可实现人物和子弹的动态移动。

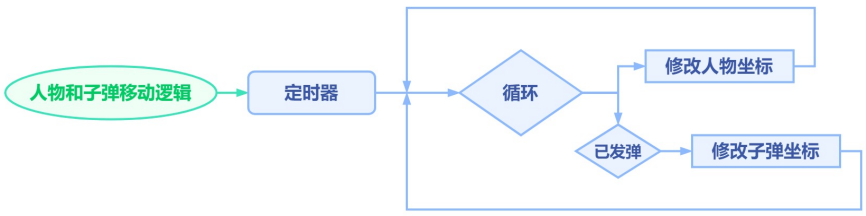

图 3-5 人物和子弹移动逻辑示意图

3.2.4 道具功能逻辑

对于两个人物击打道具的判定逻辑都是相同的，只是左边人物的子弹判断的界限是障碍物的左边界，右边人物的子弹判断的界限是障碍物的右边界，所以这里以左边人物为例，介绍击打到不同障碍物时产生的不同效果以及对其的判定逻辑。

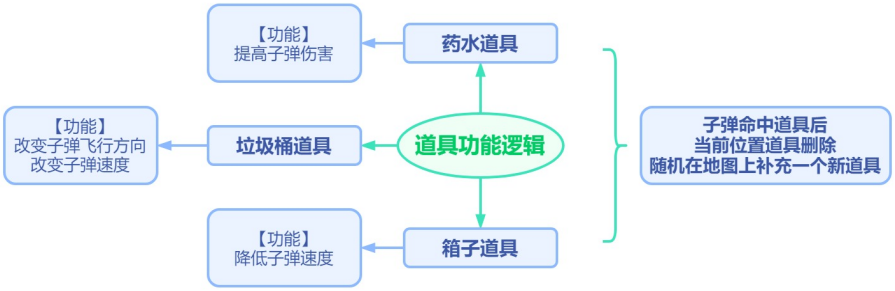

图 3-6 道具功能逻辑示意图

（1）垃圾桶道具

垃圾桶道具可以实现改变子弹的线路以及速度的功能。对于子弹运行速度的计算，是用 x 方向的方向标签值乘以 x 方向的速度，再加上 y 方向的方向标签值乘以 y 方向的速度得到的。垃圾桶道具的纵坐标被分为了上中下三个部分，若击打到上半部分，则子弹的 y 方向标签值设为 1，并把此时子弹 x 方向的速度值直接赋给子弹 y 方向的速度，从而可以实现子弹以斜向上 45 度的方向从该道具射出；若击打到中间部分，则子弹的 x 方向标签值设为-1，即实现了子弹的反弹，子弹有一定概率击打到自己；若击打到下半部分，则子弹的 y 方向标签值设为-1，并把此时子弹 x 方向的速度值直接赋给子弹 y 方向的速度，从而可以实现子弹以斜向下 45 度的方向从该道具射出。同时垃圾桶道具的功能也支持叠加，从而实现了子弹方向与速度的不断变换。

```
1    .if edx>=bound_left && edx<=bound_right && ebx<=bound_up &&  ebx>=bound_down
     ;x在范围内
2            .if ebx>=bound_m_up && ebx<=bound_up;上半部 ;speed_y+=speed_x dir_y
     =1
3                mov bullet1.dir_y,1
4                mov eax,bullet1.speed_x
5                add eax,bullet1.speed_y
6                mov bullet1.speed_y,eax
7            .endif
```

```
            .if ebx>=bound_m_down && ebx<=bound_m_up;中间反弹 dir_x反向
                mov eax,-1
                imul bullet1.dir_x
                mov bullet1.dir_x,eax
            .endif
            .if ebx<=bound_m_down && ebx>=bound_down  ;下面反弹
                mov bullet1.dir_y,-1
                mov eax,bullet1.speed_x
                add eax,bullet1.speed_y
                mov bullet1.speed_y,eax
            .endif
            ;重新更新位置
            invoke getRand,370;150-600为人物x坐标
            add eax,190;范围 190-560
            mov (Prop ptr Prop_Bounce[edi]).pos_x,eax

            invoke getRand,200;160-400为人物上下界
            add eax,180;范围 180-580
            mov (Prop ptr Prop_Bounce[edi]).pos_y,eax
            ;invoke printf,offset coord,eax,eax
    .endif
```

（2）箱子道具

箱子道具可以实现子弹的慢速。当子弹射中箱子道具时，子弹 x 方向的速度值会减 4，且该效果可以叠加。为了防止子弹速度减为 0，当速度值减小到 0 以下后，会被设置为速度的最低阈值。

```
    .if edx>=bound_left && edx<=bound_right && ebx<=bound_up &&  ebx>=bound_down
    ;x在范围内
            sub bullet1.speed_x,4
            .if bullet1.speed_x <= 0
                mov bullet1.speed_x,2
            .endif

            ;重新更新位置
            invoke getRand,370;150-600为人物x坐标
            add eax,190;范围 190-560
            mov (Prop ptr Prop_Slow[edi]).pos_x,eax

```

```
            invoke getRand,200;160-400为人物上下界
            add eax,180;范围 180-580
            mov (Prop ptr Prop_Slow[edi]).pos_y,eax
            ;invoke printf,offset coord,eax,eax
    .endif
```

（3）药水道具

药水道具可以实现子弹的威力增强。普通子弹会对人物造成一滴血的伤害，但若子弹在运行过程中碰到了药水道具，则子弹功能增强，当打到人物时会造成两滴血的伤害，且药水道具的作用可以叠加。如果子弹射中了药水道具，则此时击打到药水道具的标志值就会加 1。如果该子弹击中了人物，则人物血量的变化会根据该标志值改变，如果标记值为 0，则人物血量减 1；如果标记值大于 0，则每次人物血量减 2，标志值减 1，循环直至标记值为 0，人物血量判定结束。

```
    .if edx>=bound_left && edx<=bound_right && ebx<=bound_up &&  ebx>=bound_down
    ;x在范围内
            inc Big_flag1
            ;重新更新位置
            invoke getRand,370;150-600为人物x坐标
            add eax,190;范围 190-560
            mov (Prop ptr Prop_Big[edi]).pos_x,eax

            invoke getRand,200;160-400为人物上下界
            add eax,180;范围 180-580
            mov (Prop ptr Prop_Big[edi]).pos_y,eax
            ;invoke printf,offset coord,eax,eax
    .endif
```

当子弹射击到任意一个道具后，该道具都会消失，然后会再产生随机数，确定更新道具的位置，消失道具与更新道具的类型是相同的。

3.2.5 子弹射中判定逻辑

（1）子弹射中人物判定

根据游戏机制，人物只能由对方的子弹射中。所以我们在设计 check_hit_person 子程序的时候传入了三个参数，分别是子弹的 x，y 坐标和子弹的归属。对于已发射的子弹而言，我们需要判断子弹的位置是否与人物位置重合，若重合则判定中弹。在中弹的逻辑内我们将 person2.is_hit 标志变量设为 10，表示人物中弹的状态显示 10 帧。然后判断子弹是否为加强弹，根据这个更改子弹的伤害值。

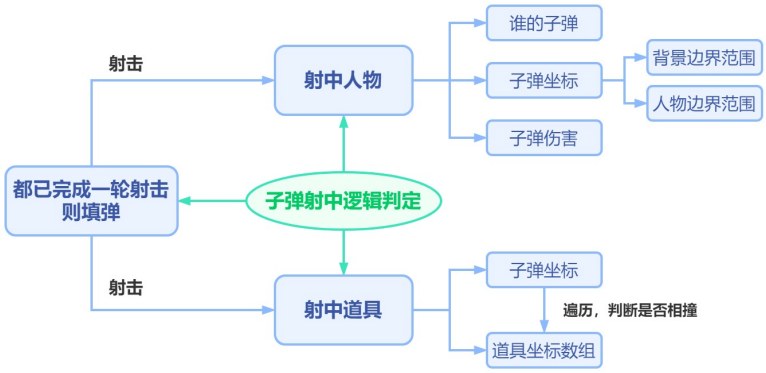

图 3-7 子弹射中判定逻辑

```
.if p==1
mov eax,person2.pos_y
sub eax,30
mov bound_down,eax
add eax,60
mov bound_up,eax
mov eax,person2.pos_x
sub eax,30
mov bound_left,eax
add eax,60
mov bound_right,eax
.if ecx<=bound_right && ecx>=bound_left && ebx <= bound_up && ebx >=bound_down
    mov person2.is_hit,10;为了让中弹的样子显示更清楚些，显示10帧
      invoke printf,offset coord,111 ,Big_flag1
      .if  Big_flag1 !=0
        .while Big_flag1>0
          sub person2.life,2
          dec Big_flag1
          invoke printf,offset life2Test,person2.life
        .endw
      .else
        sub person2.life,1
        invoke printf,offset life2Test,person2.life
      .endif
    .if person2.life == Zero || person2.life == -1 || person2.life == -2 ||
 person2.life == -3 || person2.life == -4 || person2.life == -5 ||person2.life
  == -6 ||person2.life == -7 ||person2.life == -8 ||person2.life == -9 ||
```

```
    person2.life == -10
        invoke printf,offset tip,person2.life
        mov person2.life,0
        ;invoke game_over
      .endif
      ;填弹逻辑代码在下面展示
            ...
  .endif
  .endif
```

对于子弹射出边界，我们根据位置判定若为上下界则反弹，若为左右界则消失。具体的代码实现如下：

```
  .if ecx>= 800 || ecx<=0
    mov bullet1.show,0
    .if person2.bullet == 0 && bullet1.show == 0 && bullet2.show == 0
      mov person1.bullet,1
      mov person2.bullet,1
    .endif
  .endif
  .if (ebx >400 || ebx <160)
      mov eax,-1
      imul bullet1.dir_y
      mov bullet1.dir_y,eax
  .endif
```

在射击完子弹后，我们需要判定是否需要填弹，具体的实现逻辑是若玩家 1 的子弹完成作用时玩家 2 的子弹夹为 0，则两者全部填弹；反之则不做操作。具体的代码实现如下：

```
    mov bullet1.show,0
    .if person2.bullet == 0 && bullet1.show == 0 && bullet2.show == 0
        mov person1.bullet,1
        mov person2.bullet,1
    .endif
```

（2）子弹射中道具判定

对于子弹射中道具，我们需要获取到子弹的位置，然后去遍历道具数组来判定是否与道具位置相撞。若相撞则根据道具功能来对子弹加以相关特性。这里的判定逻辑不做赘述。遍历道具数组的代码实现如下：

```
local bound_up:dword  ;up-m_up子弹向上弹
local bound_m_up:dword  ;m_up-m_down 反弹
local bound_m_down:dword;m_down-down 向下弹
local bound_down:dword

local bound_left:dword
local bound_right:dword
.if p==1;1发的子弹
  ;edx循环计数, ecx,ebx xy坐标
  mov edi,0
  mov ecx,0
  mov ebx,y
  mov edx,x
  ;Bounce
  .while ecx<Bounce_Num
    ;道具效果已展示, 不再赘述
    ...
    add edi,type Prop ;Prop大小
    inc ecx
  .endw
```

3.2.6 事件响应逻辑

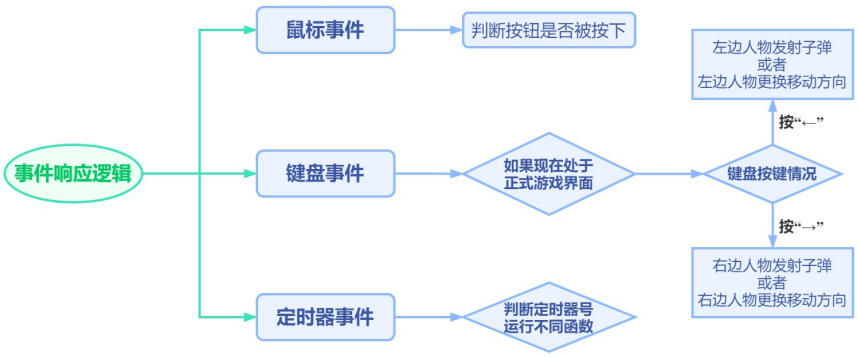

图 3-8 事件响应逻辑示意图

（1）鼠标事件

单击按钮时，需要判断出当前有鼠标点击，且鼠标点击的位置是否为按钮图片所在的范围。因此使用 iface_mouseEvent 和 judge_area 子程序来完成按钮的点击效

果，实现页面跳转。子程序 iface_mouseEvent 是用来判断鼠标是否进行了点击操作的。如果鼠标左键按下，则调用 judge_area 子程序对鼠标的点击范围进行判断；在 judge_area 子程序中，通过对比 start_button 中的范围（即“Start”的范围）和鼠标的坐标大小关系，来确定鼠标坐标是否在“Start”的图片范围之内，如果在的话，就给 eax 寄存器存值 1，否则存 0。回归 iface_mouseEvent 判断分支，判断 eax 的值，如果 eax 为 1，说明玩家已经按下“Start”，则通过 game_init 子程序完成正式游戏界面的绘制。并且打开定时器 0，并以 30ms 的时间间隔调用计时器 0 中的函数，实现游戏界面中基本循环的运行。

```
iface_mouseEvent proc C x:dword,y:dword,button:dword,event:dword
    .if button == LEFT_BUTTON && event == BUTTON_DOWN
        .if curWindow == 0;当前在主界面
            invoke judge_area,x,y,start_button.left,start_button.right,
start_button.top,start_button.bottom
            .if eax==1
                invoke printf,offset coord,x,y
                ;游戏开始，设置初始值
                invoke game_init
                ;开启循环，30ms触发
                invoke startTimer,0,30
            .endif
            invoke judge_area,x,y,exit_button.left,exit_button.right,
exit_button.top,exit_button.bottom
            .if eax ==1
                invoke ExitProcess, NULL
            .endif

        .endif
        .if curWindow == 2
            invoke judge_area,x,y,exit_button.left,exit_button.right,
exit_button.top,exit_button.bottom
            .if eax ==1
                invoke ExitProcess, NULL
            .endif
        .endif
    .endif
    ret
iface_mouseEvent endp
judge_area proc C uses ebx x:dword,y:dword,left:dword,right:dword,top:dword,
bottom:dword
```

```
28          ;x,y为鼠标按下位置
29          mov eax,x
30          mov ebx,y
31          .if eax <= left || eax >=right || ebx >= bottom || ebx <= top
32              mov eax,0
33          .else
34              mov eax,1
35          .endif
36          ret
37      judge_area endp
```

如果在判断鼠标范围时发现鼠标并未在“Start”的范围中点击，则进入“判断是否为‘Exit’按钮”分支。调用 judge_area 子程序，通过对比 exit_button 中的范围（即”Exit“的范围）和鼠标的坐标大小关系，来确定鼠标是否在“Exit”的范围之内，如果在的话 eax 此时会变为 1，通过判断 eax 等于 1，来决定调用退出进程的子程序。

（2）键盘事件

如果当前 curWindow 为 1，即代表是游戏界面，则根据键盘的输入来决定接下来的操作。在该游戏中，键盘总共会有两个输入，即“←”“→”，分别代表左边人物发射子弹和右边人物发射子弹。当判断键盘输入为“←”，且该按键被按下时，首先判断当前人物（即左边人物）是否拥有子弹，如果没有子弹，则更换其运行的方向，不能进行射击行为；如果当前有子弹，则子弹清 0，得到当前人物的位置，该位置即为子弹开始显示的坐标，并给子弹设置向右运行的方向标志（即 x 方向为 1，y 方向为 0），以及子弹每个方向上的速度（即 x 方向和 y 方向）。对于子弹运行速度的计算，是用 x 方向的方向标签值乘以 x 方向的速度，再加上 y 方向的方向标签值乘以 y 方向的速度。最后再把子弹可以显示的标志设为 1，确保子弹的正确显示。

```
1   iface_keyboardEvent proc C key:dword,event:dword
2       .if curWindow == 1
3           ;invoke printf,offset coord,person1.dir,person2.dir
4           .if key == VK_LEFT && event==BUTTON_DOWN;按下左键，！！注意一定要是
    按下;不然算两次
5               .if person1.bullet==1;有子弹
6                   mov person1.bullet,0
7                   ;其他子弹操作：设置子弹初始化信息
8                   mov eax,person1.pos_x
9                   mov bullet1.pos_x,eax
10                  mov eax,person1.pos_y
11                  mov bullet1.pos_y,eax
```

```
                    mov bullet1.dir_x,1
                    mov bullet1.dir_y,0
                    mov bullet1.speed_x,12
                    mov bullet1.speed_y,12
                    mov bullet1.show,1
                .else ;换方向
                    mov eax,-1
                    imul person1.dir
                    mov person1.dir,eax
                .endif
            .endif
            ...
        .endif
        ret
    iface_keyboardEvent endp
```

（3）定时器事件

注册了定时器事件后，便会根据定时器号选择运行不同的函数。在该游戏中，即当定时器号为 0 时，则会调用 draw_game 的函数，开始游戏界面的绘制。并且根据打开定时器 0 时对应函数语句的设置，每隔 30ms 就会调用一次该定时器事件。

```
    iface_timerEvent proc c tid: dword
        .if tid == 0;游戏主循环
            invoke draw/_game
        .endif
        ret
    iface_timerEvent endp
```

3.3 技术亮点

在本游戏项目中，我们的程序拥有以下技术亮点：

- 使用随机位置算法：我们通过自定义一个 seed 来在 getRand 中更新生成的随机数，然后用于位置的坐标中，实现位置的随机化。
- 道具的功能多样化，分别具有改变子弹飞行方向、改变子弹速度、改变子弹伤害等效果，这些效果的算法由我们自己编写及实现。
- 实现动画效果：通过人眼视觉暂留的原理，对图片的坐标位置和切换逻辑按帧进行规定，使其按帧发生变化，从而人眼获取到的就是动态的动画效果。

- 按钮判定：我们没有采用一般的窗口编程的按钮实现和点击事件判定方法，而是通过制作一个按钮的图片，然后判定图片的范围和鼠标单击位置的坐标, 如果位置在图片上，则说明点到了此按钮，进入按钮点击后的对应逻辑。
- 自定义图片和音乐：本游戏的部分图片素材由我们根据《Among us》游戏的素材自己二次制作而成，另一部分图片素材由我们自己使用 Photoshop 制作，图片和音乐的风格都符合游戏的风格特色。

第 4 章 玩家手册

4.1 运行和卸载方式

针对我们的游戏，有两种方式可以运行：如果使用 VS 运行项目，则直接删除项目即可删除游戏；如果使用我们给出的安装程序运行项目，则需要使用安装程序卸载项目。

4.1.1 使用 VS 运行项目

在依照“第 2 章开发环境”中的 MSVC 工具集版本、并选择好当前项目的配置环境（如 masm32 库的位置链接等）后，使用 VS 打开本实验项目，选择上方调试-开始执行（不调试）运行项目。

4.1.2 使用安装程序运行项目

我们使用 VS 自带的 Setup 扩展内容完成了整体项目的打包发布。打包后的压缩包名称为“Debug.zip”，解压该文件后，会生成以下文件：setup.exe 可以完成项目软件的安装；.msi 文件可以完成对软件的卸载以及修复。

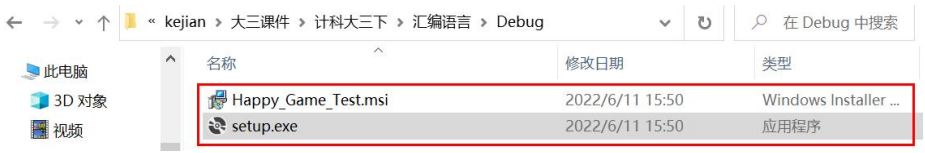

图 4-1 解压以后的文件

双击 setup.exe，会弹出图 4-2，单击“下一步”：

自定义程序所在文件夹，选择“任何人”，单击“下一步”。

单击“下一步”，进行安装。

完成安装。

在图 4-3 中自定义的文件路径里，可以找到以下文件：

双击“LocalTest.exe”后即可运行。

在完成安装后，桌面还会生成一个快捷方式，双击快捷方式也可以直接运行游戏。

4.1.3 使用安装程序卸载游戏

在图 4-1 中，双击“.msi”格式结尾的文件，弹出图 4-8。

选中“删除 Happy_Game_Test(M)”，然后单击“完成”。

图 4-2 安装页面

图 4-3 选择安装路径页面

图 4-4 确认安装

图 4-5 安装完成

图 4-6 生成的可执行文件

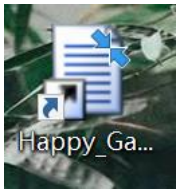

图 4-7 快捷方式

图 4-8 使用安装程序删除本地已安装游戏

在删除成功之后，单击“关闭”关闭窗口就完成了整个删除流程。

图 4-9 删除成功

4.2 运行效果与功能说明

进入游戏后我们可以看到游戏的开始界面，其中包括 Strat 和 Exit 两个选项；点击 Start 我们就进入了正式游戏页面，点击 Exit 则会退出程序运行，关闭游戏。

在游戏页面我们可以看到左右两边有两个自己移动的人物，此时游戏已经开始。游戏底部“VS”板上显示的气球代表两个玩家操作人物的生命值，在人物被击中之后生命值会依据子弹的伤害减少对应的值。

操作方式：按下键盘上的“□”可以让左边的人物发弹/改变移动方向；按下键盘上的“→”可以让右边的人物发弹/改变移动方向。

游戏规则：

- 玩家每轮拥有一发子弹，在发出子弹且未装弹时，玩家只能通过“←”或“→”改变人物的移动方向；

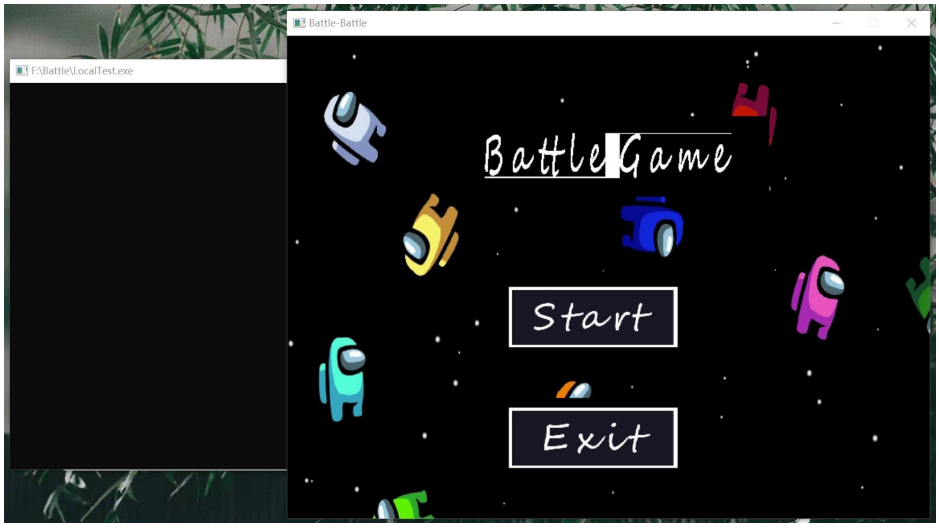

图 4-10 游戏主菜单界面

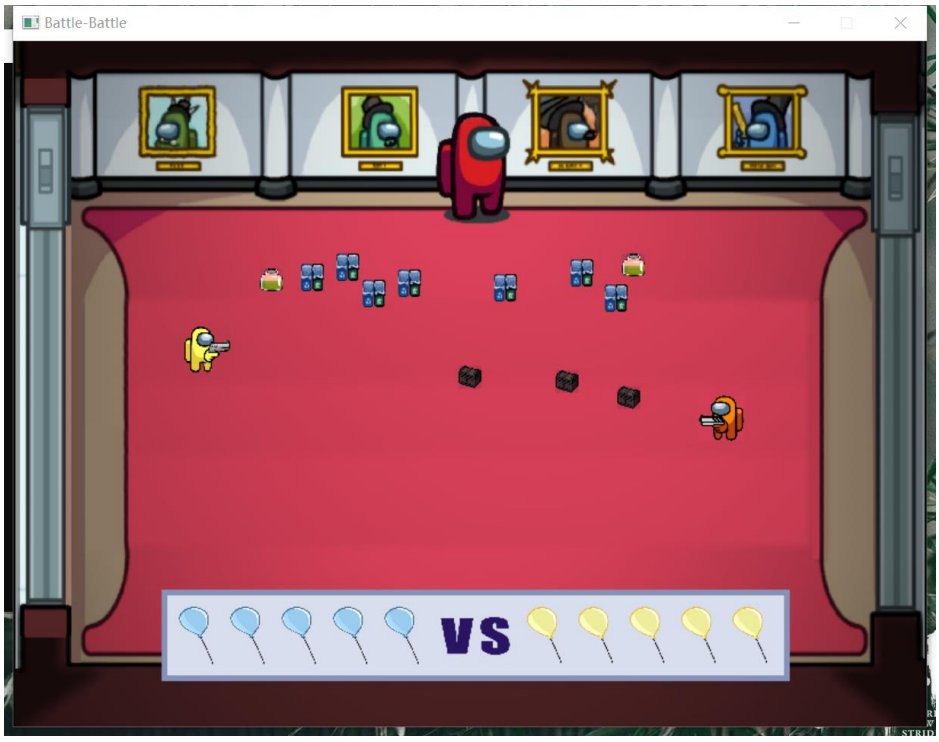

图 4-11 正式游戏界面

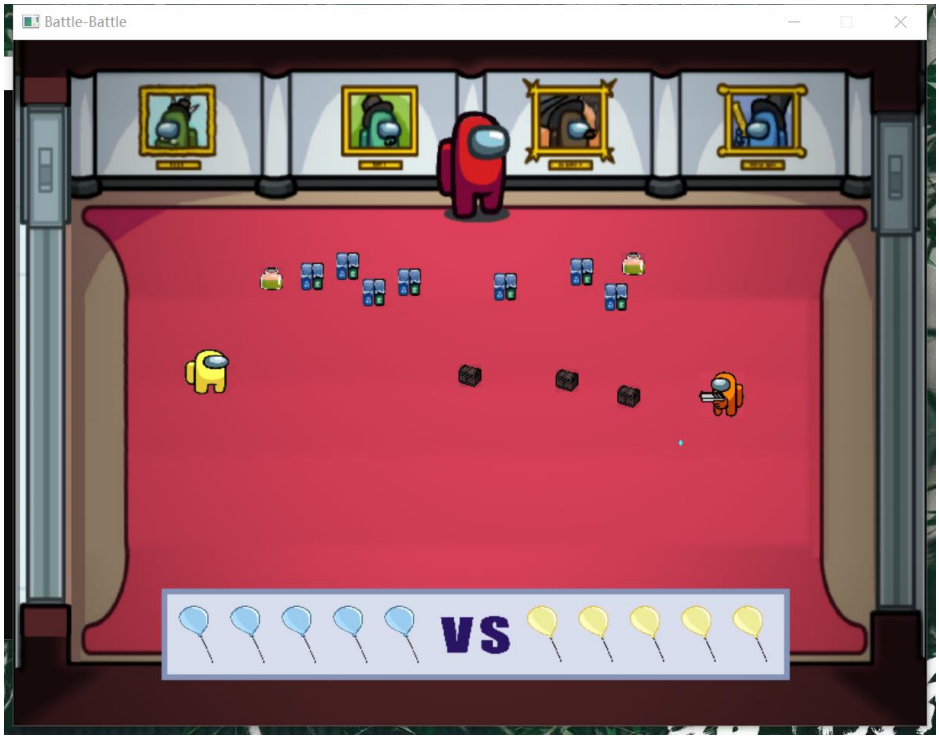

图 4-12 Player1（黄色人物）发射子弹

- 当两名玩家均完成该轮射击后，双方同时填弹进入下一轮，填弹需要花费一定的时间。
- 在界面中央会随机出现若干道具，这些道具将给已发射的子弹提供属性加成，道具的功能为：
  - 垃圾桶道具：当子弹射击到垃圾桶时，子弹会被垃圾桶弹射从而改变弹道方向；
  - 药水道具：当子弹射击到药水道具时，子弹会得到增强，伤害提高至两点；
  - 箱子道具：当子弹射击到箱子道具时，子弹的移动速度会降低，变化的子弹速度将影响另一位玩家的游戏节奏，从而提升对方躲避子弹的难度；

当某一方的生命值小于等于 0 时，自动进入到游戏结算界面，游戏结束。
此时单击“Exit”即可关闭程序运行，退出游戏。

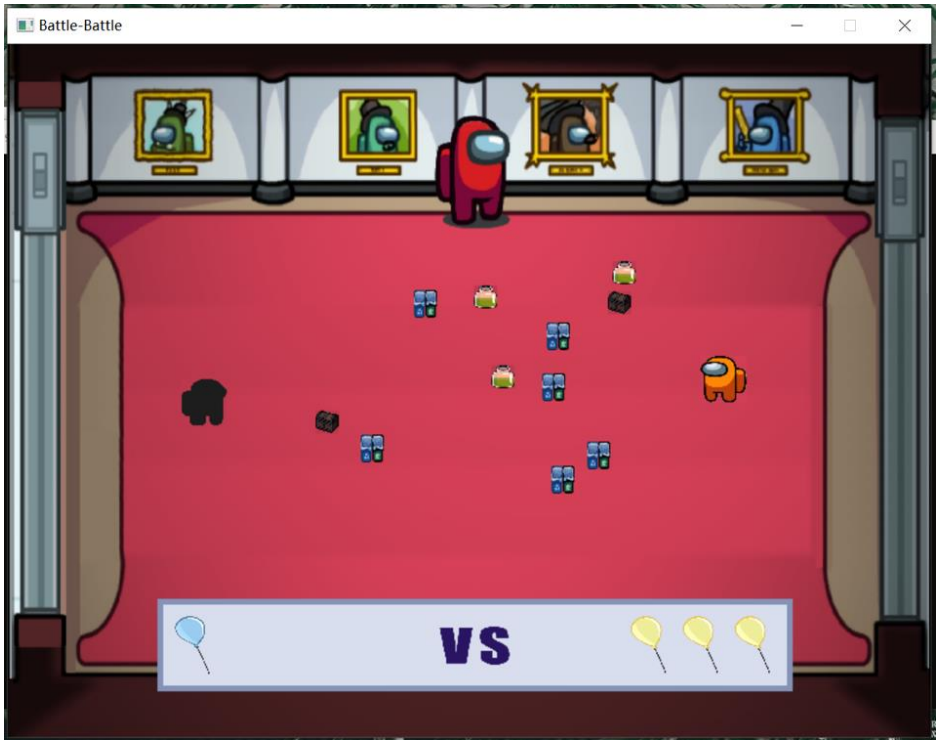

图 4-13 Player1 中弹的效果图

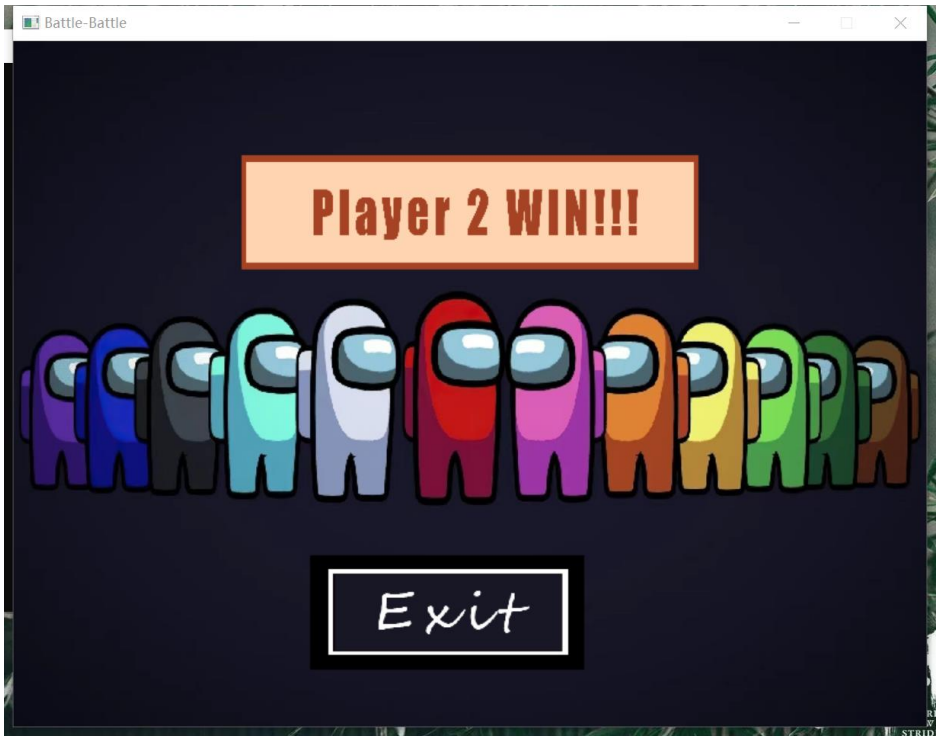

图 4-14 Player2 的获胜场景

第 5 章 总结心得

5.1 简要开发节点

```
5.20：开会确定了游戏内容、结构以及功能，并且在Github创建项目仓库；完成了前端素材的制作和上传；
5.30：完成了第一版结构的搭建以及前端界面的实现；
5.31：实现游戏人物的动态移动、子弹的动态移动以及鼠标键盘响应事件的；
6.1：完成了项目的重构；实现了中弹功能；
6.2：完成了人物状态改变、生命值功能以及发弹填弹机制；
6.5：实现了道具位置随机的效果；
6.10：完成了三种道具的功能实现；
6.11：完成击中机制的补充、结算界面的跳转以及一些bug的修复；完成项目文档的编写和代码的整理。
```

5.2 小组人员分工

王岩：完成了整体所用数据结构的完整定义，进行代码结构的二次重构以及确定，基本运行逻辑的确定，人物移动，子弹击中、碰撞的初始功能的完成，合作完成了道具随机初始化、击中判定等功能。

杨汶锦：完成了人物状态改变、发弹填弹机制、生命值逻辑判定和生命条界面显示的代码实现；参与了人物中弹的实现；完成了项目的打包以及 Latex 报告格式规范。

吴一凡：完成了代码整体框架的初次编写，实现了图片和音乐的载入，以及所有界面绘制逻辑，跳转逻辑，按钮逻辑等的编写，合作完成了子弹碰撞，道具随机初始化等功能。

蒙思洁：完成了游戏音乐素材的寻找和图片素材的制作，参与了部分界面绘制逻辑、部分跳转逻辑的撰写，修正了游戏结束判定的逻辑，完成实验报告的框架设计和分解后的模块图绘制。

5.3 小组成员心得

5.3.1 王岩

用汇编来从头写游戏还是十分熬人的，虽然有前人的架构和好用的库，但是真的从头到尾确定数据结构、架构、逻辑过程等等，到一个个细节：理解一个个子函数的作用、一些复杂数据结构如汇编里结构体的调用等等还是非常掉头发的，以及对

出其不意的 debug 还是依靠大家通力合作来解决。

考试时间紧迫，没能对所有功能做完整实现，还是有点遗憾。但是整体实验收获了许许多多的编程经验，对于汇编语言的应用也更加熟练，收获也是很多的。

5.3.2 杨汶锦

由于时间原因，汇编上机作业的小组项目和个人项目几乎是并行的。在两者间不断学习汇编的语法知识、指令使用的细节。在本次小组作业中我们清晰地感受到了将实际需求一步步细分、转化逻辑实现并最终转化代码实现的过程，在项目的各个部分间寻求协调并在一次次调试中寻找逻辑不严谨的地方，整体的开发过程虽然还算顺利但还是令人影响深刻的。

最终我们基本实现了构思初期对这个游戏的绝大部分功能，也对于汇编语言、汇编程序有了更加深刻的认识。在 debug 时对于一条条指令追本溯源地思考、对于自身逻辑地一遍遍验证，最终实现了这个游戏。感谢默契且给力的队友们！

5.3.3 吴一凡

通过此次汇编小组实验，我对于汇编语言程序设计有了更进一步的认知，学习到了如何使用汇编语言完成一个较大规模程序的编写。在实验初期，由于对于汇编语言结构的不熟悉以及对于代码整体框架的不了解，不知道应该如何组织好整个程序，但在查找了相关资料，并且不断尝试后，我们最终逐步搭起了整个程序的框架。同时由于对一些库函数的不了解，所以在代码编写初期，经常需要查找相关资料，学习如何正确的调用函数。

同样，在整个代码编写过程中我们也遇到了许多问题，也在不断对游戏逻辑进行完善。例如图片无法正确显示，循环无法正确调用，子弹行进路径与预期不符，人物属性出现错误等等。但这些问题最终也都在不断改进，小组成员互帮互助下得以解决。完成了此次实验后，我也收获了许多经验，提高了自己汇编代码的编写能力。

5.3.4 蒙思洁

此次汇编小组的实验令我对汇编语言的了解更深一层，三个个人的小实验令我学到了汇编语言基础的语句操作，而本小组实验令我对汇编语言的开发过程有了更深的认知。通过对整体游戏的效果想象，我们能够首先绘制出游戏每个界面需要什么素材、控件，绘制出主题游戏部分每个素材它的作用是什么，设计它的实现方法；在游戏开发初期需要对素材进行寻找和处理，使其能够合适地被使用，在游戏开发中期需要划分逻辑结构，弄明白每个功能的实现需要哪几方面的共同作用；在游戏开发末期需要对游戏进行大量覆盖性的测试和修正，同时要对应初期设计好的目标来完善那些未完成的功能。

在代码编写中，问题是不可避免的。不论是刚开始搭建框架代码时，基础的图片素材无法被调用、小组成员的环境配置版本不一带来的无法运行的 bug；还是后期在测试过程中发现，道具没有正常发挥作用、子弹停留在空中不继续移动的问题、人物的生命值降为 0 以后不能正确弹出 game_over 界面等问题。但是只要肯去钻研，在一行一行代码中捋逻辑，在一个一个重复出现的变量名称中判断它的变化情况，在通力合作之下，我们最终总是能解决这些问题。在这次实验中，队友们的积极沟通和高效合作是我们小组的游戏程序能够完成开发的最重要的组成部分！

5.4 不足和展望

整个项目借鉴了《微观战争》游戏的一部分游戏机制以及操作机制，我们在复现其机制的基础上设计了三种道具特效。

由于整体的项目开发时间短，小组成员精力有限，游戏仍存在以下几方面的不足：

- 整体项目采用单线程设计，在高并发操作的情况下可能出现卡顿、崩溃的情况；
- 游戏整体动效不够平滑，对于动画关键帧的选取不够准确，游戏看起来较为简单廉价；
- 游戏开发过程中的协同不够智能高效，未高效使用 git 等版本控制技术提高协同开发效率；

对于这个游戏，我们还有以下可实现的提高和展望：

- 采用多线程重构游戏：由于本次游戏涉及到键盘输入、逻辑计算和动画实现，当出现大量计算或高频次并发键盘输入时，游戏可能会出现卡顿和崩溃的可能性；我们可以采用多线程的方式通过设置全局的标志变量来进行多个线程间的通信，从而提高游戏的鲁棒性；
- 补充游戏机制：还可以为游戏增加更加多元化的玩法和彩蛋。比如说，可以在正式游戏界面增加菜单功能，在菜单里可以调整游戏的进度和音乐等效果；可以在玩家中弹以后增加一些成就系统。

5.5 写在最后

本次小组项目的实现离不开小组成员的付出和钻研、离不开张全新老师课堂上细致认真的讲授、离不开在 GitHub、Google 贡献汇编开发技巧和问题解决方案的每一位前辈。感谢各位有形或无形的帮助，祝好！