南開大學
Nankai University

南开大学

密码与网络空间安全学院

大作业实验报告

宝可梦主题碰撞小游戏

姓名：杨滢

学号：2412266

专业：信息安全、法学双学位

2025 年 5 月 25 日

目录

1 实验题目

宝可梦主题碰撞小游戏

2 开发环境

Visual Studio 2022、EasyX 图形库

3 实验要求

- 基于面向对象的编程思想进行项目设计
- 利用 EasyX 图形库实现游戏界面的可视化
- 利用 mciSendString 实现游戏 bgm 的播放与停止
- 实现角色选择、移动控制、敌人生成与移动、碰撞检测、游戏状态管理等游戏功能模块的开发

4 github 仓库

本次实验的有关代码和文件，都已经上传至我的个人 github 中。您可以通过访问**此链接**来查阅我的代码文件。

5 实验流程

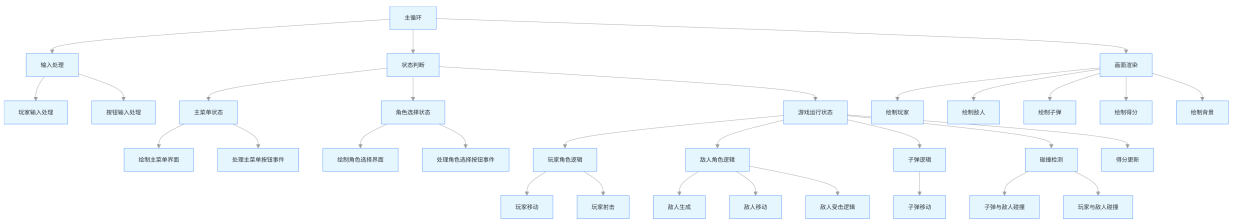

图 5.1: 整体架构实现思路

5.1 实现动画帧的加载与连续播放

这里主要通过 *Atlas* 类和 *Animation* 类来实现。

Atlas 类负责管理游戏中的动画帧集合，它将一系列相关的图片资源加载到内存中，并组织成一个帧列表. 同时使用 vector<IMAGE*> 存储所有帧的指针，在析构时释放资源，避免内存泄漏。

```
class Atlas {
public:
  Atlas(LPCTSTR path, int num) {
    TCHAR path_file[256];
    for (size_t i = 0; i < num; i++) {
```

```
        _stprintf_s(path_file, path, i);
        IMAGE* frame = new IMAGE();
        loadimage(frame, path_file);
      frame_list.push_back(frame);
      }
   }
   ~Atlas() {
      for (size_t i = 0; i < frame_list.size(); i++) {
        delete frame_list[i];
      }
   }
public:
   std::vector<IMAGE*>frame_list;
};
```

Animation 类负责控制动画的播放逻辑，包括帧切换、播放速度和绘制位置。

```
class Animation {
public:
   Animation(Atlas* atlas, int interval) {
      anim_atlas = atlas;
      interval_ms = interval;
   }
   ~Animation() = default;
   void Play(int x, int y, int delta) {
      timer += delta;
      if (timer >= interval_ms) {
        idx_frame = (idx_frame + 1) % anim_atlas->frame_list.size();
        timer = 0;
      }
      putimage_alpha(x, y, anim_atlas->frame_list[idx_frame]);
   }
private:
   int timer = 0;
   int idx_frame = 0;
   int interval_ms = 0;
   Atlas* anim_atlas;
};
```

5.2 角色系统的设计

这里主要包含玩家 Player 类和敌人 Enemy 类。

5.2.1 玩家类

- 属性：拥有位置 position、移动状态 is_move_up is_move_down is_move_left is_move_right、得分 score、玩家角色的宽度 $PLAYER\_WIDTH$ 和高度 $PLAYER\_HEIGHT$ 等属性。
- 行为：通过 ProcessEvent 函数处理键盘输入，改变移动状态 Move 函数根据移动状态更新位置，并进行边界检测，防止角色超出游戏界面。Draw 函数根据角色的移动方向选择相应的动画（$anim\_left$ 或 $anim\_right$）进行绘制，并且绘制角色的影子。

5.2.2 敌人类

- 属性：拥有位置 position、存活状态 alive、移动方向 $facing\_left$、宽度 $ENEMY\_WIDTH$ 和高度 $ENEMY\_HEIGHT$ 等属性。
- 行为：在构造函数中，根据随机生成的边缘位置 SpawnEdge 初始化敌人的位置。Move 函数使敌人向玩家位置移动，Draw 函数根据敌人的移动方向选择相应的动画 $anim\_left$ 或 $anim\_right$ 进行绘制，并绘制敌人的影子。CheckBulletCollision 函数检测子弹与敌人的碰撞；CheckPlayerCollision 函数检测敌人与玩家的碰撞；Hurt 函数处理敌人受到伤害的逻辑，将敌人的存活状态设为 false。

5.3 UI 系统的设计

UI 系统是游戏或应用程序中负责与用户进行交互的视觉组件集合。它的核心作用是将程序功能转化为用户可感知和操作的可视化界面，并处理用户输入以触发相应功能。

5.3.1 按钮类

这里采用了 Button 类的**继承和重载**。我的代码通过继承实现了 StartGameButton、QuitGameButton、CharacterButton 三种按钮。其继承关系如下图所示：

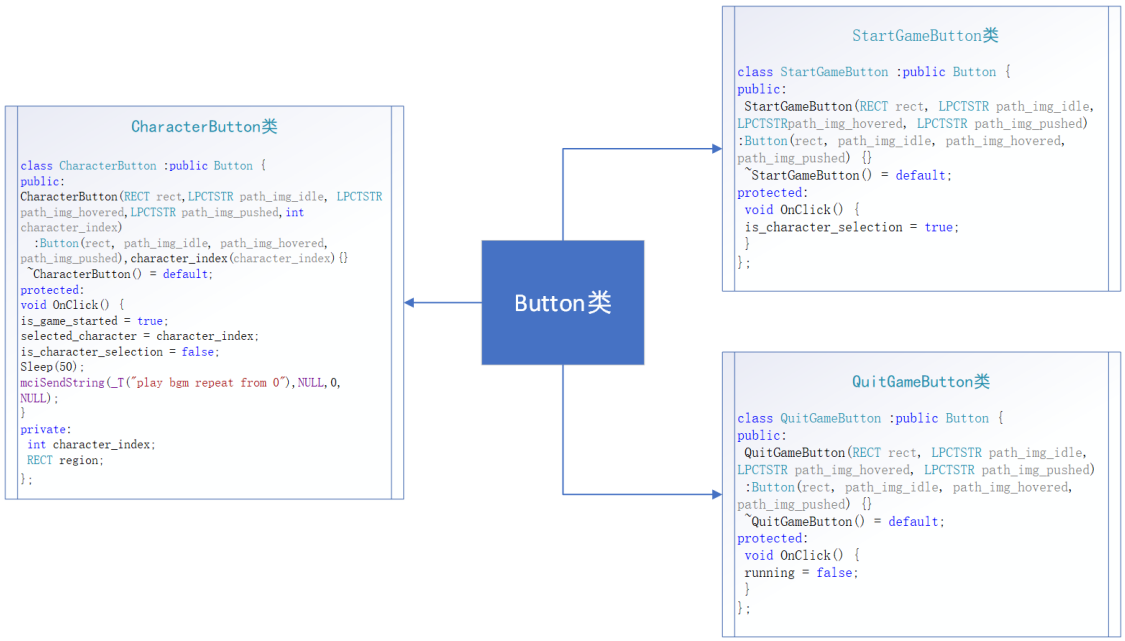

图 5.2: button 类图

5.3.2 界面管理

我的游戏包含主菜单、角色选择界面、游戏主界面这三个界面：

- 主菜单主要是游戏开始和结束按钮，实现向角色选择界面的切换和游戏的退出。
- 角色选择界面要显示三个角色选择按钮，完成角色选择功能并向游戏主界面切换。
- 游戏主界面要实现游戏主要功能，并且显示玩家、敌人、子弹和得分。

对于不同界面的转换，主要依赖以下三个状态变量控制游戏流程

- *running* 控制游戏主循环
- *is_game_started* 标识游戏是否开始
- *is_character_selection* 标识是否处于角色选择界面

在主循环中，通过分支语句和这三个状态变量，实现界面之间的切换。

```
while (peekmessage(&msg)) {
  if (is_game_started) {
     player->ProcessEvent(msg);
  }
  else if (is_character_selection) {
     DrawCharacterSelection();
  }
  else {
     btn_start_game.ProcessEvent(msg);
     btn_quit_game.ProcessEvent(msg);
  }
}
```

5.3.3 视觉管理

视觉管理部分，本游戏主要通过 EasyX 图形库实现可视化，主要包含了：

- 绘制透明底图片的函数

```
inline void putimage_alpha(int x, int y, IMAGE* img) {
   int w = img->getwidth();
   int h = img->getheight();
   AlphaBlend(GetImageHDC(NULL), x, y, w, h,
      GetImageHDC(img), 0, 0, w, h, { AC_SRC_OVER,0,255,AC_SRC_ALPHA });
}
```

- 插入游戏背景图片

```
IMAGE img_background;
putimage(0, 0, &img_character_bg);
```

- 游戏结束的分数弹窗

```
if (is_game_over) {
    TCHAR text[128];
    _stprintf_s(text, _T("最终得分为：%d! \n点击确定返回主菜单"), score);
    mciSendString(_T("stop bgm"), NULL, 0, NULL);
    if (MessageBox(GetHWnd(), text, _T("游戏结束"), MB_OKCANCEL |
        MB_ICONINFORMATION) == IDOK) {
        is_game_started = false;
        is_character_selection = false;
        score = 0;
        player->position = { 300, 300 };
        // 释放资源并重置游戏状态，此部分代码在此处省略
    }
}
```

5.3.4 听觉管理

听觉管理部分，本游戏主要通过 Windows Multimedia API(mciSendString) 来实现效果：

- 在游戏初始化阶段，使用 mciSendString 加载并播放背景音乐

```
mciSendString(_T("open mus/bgm.mp3 alias bgm"), NULL, 0, NULL);
```

- 在游戏结束时停止 BGM

```
if (is_game_over) {
    mciSendString(_T("stop bgm"), NULL, 0, NULL);// ...
}
```

5.4 主要游戏功能的实现：碰撞检测

这里采取圆形碰撞检测。主要思想是将环绕在玩家周围的子弹等效为一个点，看其是否在敌人的圆形中；把敌人也等效为点，来判断其是否在玩家圆形内。

```
bool CheckBulletCollision(const Bullet& bullet) {
  // 计算敌人中心点
  float enemy_center_x = position.x + ENEMY_WIDTH / 2;
  float enemy_center_y = position.y + ENEMY_HEIGHT / 2;

```

```
    // 计算子弹到敌人中心的距离平方
    float dx = bullet.position.x - enemy_center_x;
    float dy = bullet.position.y - enemy_center_y;
    float distance_squared = dx * dx + dy * dy;

    // 比较距离平方与半径和的平方
    float radius_sum = ENEMY_RADIUS + bullet.GetR();
    return distance_squared <= radius_sum * radius_sum;
}

bool CheckPlayerCollision(const Player& player) {
    // 计算敌人中心点
    float enemy_center_x = position.x + ENEMY_WIDTH / 2;
    float enemy_center_y = position.y + ENEMY_HEIGHT / 2;

    // 计算玩家中心点
    float player_center_x = player.GetPosition().x + player.PLAYER_WIDTH / 2;
    float player_center_y = player.GetPosition().y + player.PLAYER_HEIGHT / 2;

    // 计算两中心点距离平方
    float dx = player_center_x - enemy_center_x;
    float dy = player_center_y - enemy_center_y;
    float distance_squared = dx * dx + dy * dy;

    // 比较距离平方与半径和的平方
    float radius_sum = ENEMY_RADIUS + player.GetR();
    return distance_squared <= radius_sum * radius_sum;
}
```

这样处理，相比矩形检测，除了准确性有所提高，还可以通过改变检测圆形的半径来便捷地调节游戏的难易程度。

```
// 在敌人类中
const float ENEMY_RADIUS = 15.0f; // 敌人圆形碰撞半径
}
```

6 实验结果展示

本次实验成功实现了包含角色移动、碰撞检测、界面交互等功能的游戏原型，基本达成了预期目标。在基础功能稳定性和视觉表现上效果良好。

6.1 主菜单

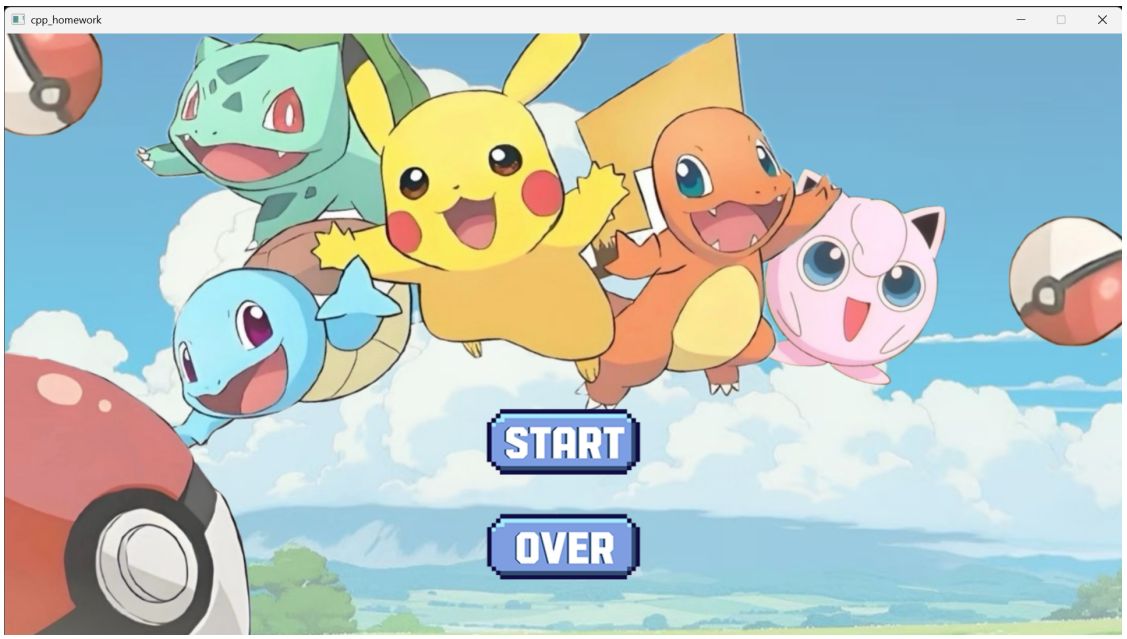

图 6.3: 主菜单

6.2 角色选择页面

图 6.4: 角色选择界面

6.3 游戏页面

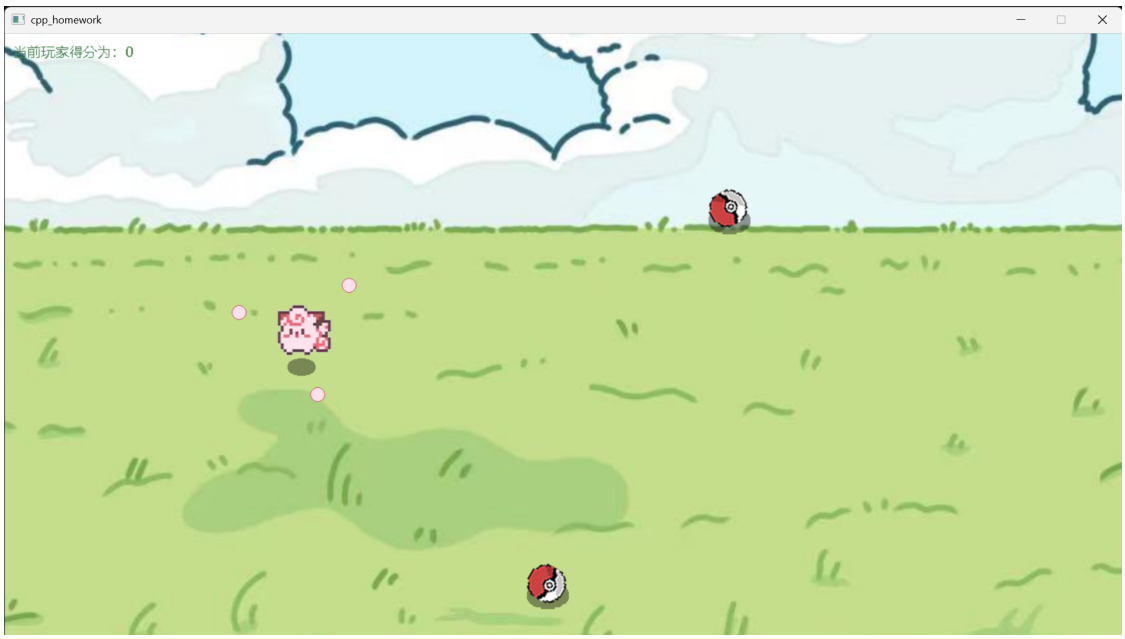

图 6.5: 游戏界面

7 实验分析

7.1 功能实现分析

- 角色移动与控制
- 动画系统实现
- 碰撞检测机制

7.2 技术实现分析

- 面向对象设计:Button 类通过虚函数 OnClick() 实现多态行为
- 资源管理: 角色类在析构时释放关联的动画资源

7.3 性能与稳定性分析

7.3.1 帧率控制

这里运用计时器控制动画帧率、敌人生成间隔和游戏帧率稳定。

1. 动画计时器 Animation::Play
   - delta 参数表示从上一帧到当前帧的时间差
   - timer 累积时间，当超过 $interval\_ms$ 时切换到下一帧
   - 通过 $interval\_ms$ 控制动画速度，其值越小动画越快

```
void Play(int x, int y, int delta) {
    timer += delta;
    if (timer >= interval_ms) {
        idx_frame = (idx_frame + 1) % anim_atlas->frame_list.size();
        timer = 0;
    }
    putimage_alpha(x, y, anim_atlas->frame_list[idx_frame]);
}
```

2. 敌人生成计时器 TryGenerateEnemy 函数
   - 每帧递增 counter，当达到 INTERVAL 时生成敌人
   - 生成间隔依赖于游戏帧率

```
void TryGenerateEnemy(std::vector<Enemy*>& enemy_list) {
   const int INTERVAL = 100;
   static int counter = 0;
   if ((++counter) % INTERVAL == 0) {
      enemy_list.push_back(new Enemy());
   }
};
```

3. 主循环帧率控制
   - 使用 GetTickCount() 获取当前时间戳
   - 计算每帧实际耗时 $delta\_time$
   - 如果耗时小于目标帧时间，则睡眠补足时间

```
// 主循环中的帧率控制
DWORD begin_time = GetTickCount();
// 游戏逻辑和渲染,此处省略
DWORD end_time = GetTickCount();
DWORD delta_time = end_time - begin_time;
if (delta_time < 1000 / 60) {
    Sleep(1000 / 60 - delta_time);
}
```

7.3.2 内存管理

利用 new 和 delete 操作实现内存管理。

8 实验总结

8.1 实验收获

- 代码能力提升:
  更好地掌握了 C++ 面向对象编程思想，在实践中加深了对类的继承、循环结构、代码内存管理、vector 使用等等的理解；同时，学会了如何运用 EasyX 库实现图形界面开发，以及如何处理键盘、鼠标事件等 Windows 消息机制。
- 问题解决能力提升:
  实验过程中遇到了很多 bug，通过各种查阅资料，反复修改试验，最终改掉了 bug。

8.2 存在问题与改进方向

- 改用更精确的碰撞检测：现在的碰撞检测采用简单的矩形重叠判断，可能存在精度不足问题
- 角色系统设计也可以尝试设计一个 character 抽象基类，Player 类和 Enemy 类对其继承与重载
- 游戏功能优化：可以增加更多角色，设计更多功能丰富游戏玩法
- 实现存档与读取功能：记录玩家历史最高得分和游戏进度

8.3 未来展望

- 学好编程语言基础，锻炼自己的代码能力
- 进一步学习编程技术，实现多人联机对战功能，提升游戏趣味性与社交性
- 学习更专业的游戏引擎（如 Unity）

9 附录：实验整体代码（带注释）

```
#include<easyx.h>
#include<vector>
#include<cmath>

IMAGE img_background;
IMAGE img_shadow;

#pragma comment(lib,"MSIMG32.LIB")
#pragma comment(lib,"Winmm.lib")


bool running = true;
bool is_game_started = false;
bool is_character_selection = false; // 角色选择状态
int selected_character = 0;      // 选中的角色索引
```

```
16
17 // 角色选择按钮布局参数
18 const int CHAR_BTN_WIDTH = 160; // 单个按钮宽度
19 const int CHAR_BTN_HEIGHT = 200; // 单个按钮高度
20 const int CHAR_BTN_SPACING = 40; // 按钮间距
21 const int CHAR_BTN_TOP = 300;   // 距离顶部的固定位置
22
23 //绘制透明底图片的自定义函数
24 inline void putimage_alpha(int x, int y, IMAGE* img) {
25    int w = img->getwidth();
26    int h = img->getheight();
27    AlphaBlend(GetImageHDC(NULL), x, y, w, h,
28       GetImageHDC(img), 0, 0, w, h, { AC_SRC_OVER,0,255,AC_SRC_ALPHA });
29 }
30
31 class Atlas {
32 public:
33    Atlas(LPCTSTR path, int num) {
34       TCHAR path_file[256];
35       for (size_t i = 0; i < num; i++) {
36          _stprintf_s(path_file, path, i);
37
38          IMAGE* frame = new IMAGE();
39          loadimage(frame, path_file);
40          frame_list.push_back(frame);//利用pushback函数把图片对象的指针添加到容器里面
41       }
42    }
43    ~Atlas() {
44       for (size_t i = 0; i < frame_list.size(); i++) {
45          delete frame_list[i];
46       }
47    }
48
49 public:
50    std::vector<IMAGE*>frame_list;
51 };
52
53 // 存储不同角色的动画图集
54 Atlas* atlas_player_left[3]; // 3个角色的左移动画
55 Atlas* atlas_player_right[3]; // 3个角色的右移动画
56 Atlas* atlas_enemy_left;
57 Atlas* atlas_enemy_right;
```

```
58
59 class Animation {
60 public:
61     Animation(Atlas* atlas, int interval) {
62         anim_atlas = atlas;
63         interval_ms = interval;
64     }
65
66     ~Animation() = default;
67
68     //动画播放
69     void Play(int x, int y, int delta) {
70         timer += delta;
71
72         if (timer >= interval_ms) {
73             idx_frame = (idx_frame + 1) % anim_atlas->frame_list.size();
74             timer = 0;
75         }
76
77         putimage_alpha(x, y, anim_atlas->frame_list[idx_frame]);
78     }
79
80
81 private:
82     int timer = 0;//动画计时器
83     int idx_frame = 0;//动画帧索引
84     int interval_ms = 0;
85     Atlas* anim_atlas;
86 };
87
88 const int WIDTH0 = 1280;//页面宽度
89 const int HEIGHT0 = 720;//页面高度
90
91 const int BUTTON_WIDTH = 190;
92 const int BUTTON_HEIGHT = 75;
93
94 class Player {
95 public:
96     const int PLAYER_WIDTH = 80;//玩家宽度
97     const int PLAYER_HEIGHT = 80;//玩家高度
98     const float PLAYER_RADIUS = 20.0f; // 玩家圆形碰撞半径
99     POINT position = { 300,300 };//初始化玩家位置
```

```
100 public:
101     Player() {
102         loadimage(&img_shadow, _T("img/shadow_player.png"));
103         // 使用选中的角色图集
104         anim_left = new Animation(atlas_player_left[selected_character], 45);
105         anim_right = new Animation(atlas_player_right[selected_character], 45);
106     };
107 
108     ~Player() {
109         delete anim_left;
110         delete anim_right;
111     }
112 
113     void ProcessEvent(const ExMessage& msg) {
114         //通过按键来实现角色移动
115         if (msg.message == WM_KEYDOWN) {
116             switch (msg.vkcode) {
117             case VK_UP:
118                 is_move_up = true;
119                 break;
120             case VK_DOWN:
121                 is_move_down = true;
122                 break;
123             case VK_LEFT:
124                 is_move_left = true;
125                 break;
126             case VK_RIGHT:
127                 is_move_right = true;
128                 break;
129             }
130         }
131         else if (msg.message == WM_KEYUP) {
132             switch (msg.vkcode) {
133             case VK_UP:
134                 is_move_up = false;
135                 break;
136             case VK_DOWN:
137                 is_move_down = false;
138                 break;
139             case VK_LEFT:
140                 is_move_left = false;
141                 break;
```

```
142         case VK_RIGHT:
143             is_move_right = false;
144             break;
145         }
146     }
147 }
148 
149 void Move() {
150     //利用向量知识来解决斜方向移动速度特别快的问题
151     int dir_x = is_move_right - is_move_left;
152     int dir_y = is_move_down - is_move_up;;
153     double len_dir = sqrt(dir_x * dir_x + dir_y * dir_y);
154     if (len_dir != 0) {
155         double normalized_x = dir_x / len_dir;
156         double normalized_y = dir_y / len_dir;
157         position.x += (int)(SPEED * normalized_x);
158         position.y += (int)(SPEED * normalized_y);
159     }
160 
161     //对玩家位置进行校准，防止玩家超出游戏界面
162     if (position.x < 0)
163         position.x = 0;
164     if (position.y < 0)
165         position.y = 0;
166     if (position.x + PLAYER_WIDTH > WIDTH0)
167         position.x = WIDTH0 - PLAYER_WIDTH;
168     if (position.y + PLAYER_HEIGHT > HEIGHT0)
169         position.y = HEIGHT0 - PLAYER_HEIGHT;
170 }
171 
172 void Draw(int delta) {
173     int pos_shadow_x = position.x + (PLAYER_WIDTH / 2 - SHADOW_WIDTH / 2);
174     int pos_shadow_y = position.y + PLAYER_HEIGHT - 8;//偏移一点点
175     putimage_alpha(pos_shadow_x, pos_shadow_y, &img_shadow);
176 
177     static bool facing_left = false;
178     int dir_x = is_move_right - is_move_left;
179     if (dir_x < 0)
180         facing_left = true;
181     else if (dir_x > 0)
182         facing_left = false;
183 
```

```
184         if (facing_left)
185             anim_left->Play(position.x, position.y, delta);
186         else
187             anim_right->Play(position.x, position.y, delta);
188     }
189
190     const POINT& GetPosition() const {
191         return position;
192     };
193
194     // 获取玩家半径
195     float GetR() const {
196         return PLAYER_RADIUS;
197     }
198
199 private:
200     const int SHADOW_WIDTH = 32;//影子大小
201     const int SPEED = 5;//定义速度
202
203 private:
204     IMAGE img_shadow;
205     Animation* anim_left;
206     Animation* anim_right;
207     bool is_move_up = false;
208     bool is_move_down = false;
209     bool is_move_left = false;
210     bool is_move_right = false;
211 };
212
213 class Bullet {
214 public:
215     POINT position = { 0,0 };
216     const int R = 8; // 子弹半径
217
218 public:
219     Bullet() = default;
220     ~Bullet() = default;
221
222     void Draw() const {
223         // 根据选择的角色改变子弹颜色
224         switch (selected_character) {
225         case 0:
```

```
226         setfillcolor(RGB(252, 225, 235));
227         setlinecolor(RGB(215, 108, 124));
228         break;
229       case 1:
230         setlinecolor(RGB(134, 218, 227));
231         setfillcolor(RGB(51, 166, 223));
232         break;
233       case 2:
234         setlinecolor(RGB(250, 210, 89));
235         setfillcolor(RGB(255, 183, 43));
236         break;
237       default:
238         setlinecolor(RGB(RED);
239         setfillcolor(RED);
240       }
241       fillcircle(position.x, position.y, R);
242     }
243
244     // 获取子弹半径
245     int GetR() const {
246       return R;
247     }
248 };
249
250
251 class Enemy {
252 public:
253     //敌人生成边界
254     enum class SpawnEdge {
255       Up = 0,
256       Down,
257       Left,
258       Right
259     };
260
261     const int ENEMY_WIDTH = 60; // 敌人宽度
262     const int ENEMY_HEIGHT = 60; // 敌人高度
263     const float ENEMY_RADIUS = 15.0f; // 敌人圆形碰撞半径
264
265     Enemy() {
266       loadimage(&img_shadow, _T("img/shadow_enemy.png"));
267       anim_left = new Animation(atlas_enemy_left, 45);
```

```
268     anim_right = new Animation(atlas_enemy_right, 45);
269
270     SpawnEdge edge = (SpawnEdge)(rand() % 4);
271     switch (edge) {
272     case SpawnEdge::Up:
273       position.x = rand() % WIDTH0;
274       position.y = -ENEMY_HEIGHT;
275       break;
276     case SpawnEdge::Down:
277       position.x = rand() % WIDTH0;
278       position.y = HEIGHT0;
279       break;
280     case SpawnEdge::Left:
281       position.x = -ENEMY_WIDTH;
282       position.y = rand() % HEIGHT0;
283       break;
284     case SpawnEdge::Right:
285       position.x = WIDTH0;
286       position.y = rand() % HEIGHT0;
287       break;
288     default:
289       break;
290     }
291   }
292
293   bool CheckBulletCollision(const Bullet& bullet) {
294     // 计算敌人中心点
295     float enemy_center_x = position.x + ENEMY_WIDTH / 2;
296     float enemy_center_y = position.y + ENEMY_HEIGHT / 2;
297
298     // 计算子弹到敌人中心的距离平方
299     float dx = bullet.position.x - enemy_center_x;
300     float dy = bullet.position.y - enemy_center_y;
301     float distance_squared = dx * dx + dy * dy;
302
303     // 比较距离平方与半径和的平方
304     float radius_sum = ENEMY_RADIUS + bullet.GetR();
305     return distance_squared <= radius_sum * radius_sum;
306   }
307
308   bool CheckPlayerCollision(const Player& player) {
309     // 计算敌人中心点
```

```
    float enemy_center_x = position.x + ENEMY_WIDTH / 2;
    float enemy_center_y = position.y + ENEMY_HEIGHT / 2;

    // 计算玩家中心点
    float player_center_x = player.GetPosition().x + player.PLAYER_WIDTH / 2;
    float player_center_y = player.GetPosition().y + player.PLAYER_HEIGHT / 2;

    // 计算两中心点距离平方
    float dx = player_center_x - enemy_center_x;
    float dy = player_center_y - enemy_center_y;
    float distance_squared = dx * dx + dy * dy;

    // 比较距离平方与半径和的平方
    float radius_sum = ENEMY_RADIUS + player.GetR();
    return distance_squared <= radius_sum * radius_sum;
}

void Move(const Player& player) {
    const POINT& player_position = player.GetPosition();
    int dir_x = player_position.x - position.x;
    facing_left = (dir_x < 0);
    int dir_y = player_position.y - position.y;
    double len_dir = sqrt(dir_x * dir_x + dir_y * dir_y);
    if (len_dir != 0) {
        double normalized_x = dir_x / len_dir;
        double normalized_y = dir_y / len_dir;
        position.x += (int)(SPEED * normalized_x);
        position.y += (int)(SPEED * normalized_y);
    }
}

void Draw(int delta) {
    int pos_shadow_x = position.x + (ENEMY_WIDTH / 2 - SHADOW_WIDTH / 2);
    int pos_shadow_y = position.y + ENEMY_HEIGHT - 31;//偏移一点点
    putimage_alpha(pos_shadow_x, pos_shadow_y, &img_shadow);

    if (facing_left)
        anim_left->Play(position.x, position.y, delta);
    else
        anim_right->Play(position.x, position.y, delta);
}

```

```
352     const POINT& GetPosition() const {
353         return position;
354     }
355 
356     ~Enemy() {
357         delete anim_left;
358         delete anim_right;
359     }
360 
361     void Hurt() {
362         alive = false;
363     }
364 
365     bool CheckAlive() {
366         return alive;
367     }
368 
369 private:
370     // const int ENEMY_WIDTH = 60;//敌人宽度
371     // const int ENEMY_HEIGHT = 60;//敌人高度
372     const int SHADOW_WIDTH = 48;//影子大小
373     const int SPEED = 4;//定义速度
374 
375 private:
376     IMAGE img_shadow;
377     Animation* anim_left;
378     Animation* anim_right;
379     POINT position = { 0,0 };//初始化敌人位置;
380     bool facing_left = false;
381     bool alive = true;
382     bool is_move_up = false;
383     bool is_move_down = false;
384     bool is_move_left = false;
385     bool is_move_right = false;
386 };
387 
388 
389 
390 //按钮类
391 class Button {
392 public:
393     Button(RECT rect, LPCTSTR path_img_idle, LPCTSTR path_img_hovered, LPCTSTR
```

```
        path_img_pushed) {
394     region = rect;
395     loadimage(&img_idle, path_img_idle);
396     loadimage(&img_hovered, path_img_hovered);
397     loadimage(&img_pushed, path_img_pushed);
398   }
399
400   ~Button() = default;
401
402   void ProcessEvent(const ExMessage& msg) {
403     switch (msg.message) {
404     case WM_MOUSEMOVE:
405       if (status == Status::Idle && CheckCursorHit(msg.x, msg.y))
406         status = Status::Hovered;
407       else if (status == Status::Hovered && !CheckCursorHit(msg.x, msg.y))
408         status = Status::Idle;
409       break;
410     case WM_LBUTTONDOWN:
411       if (CheckCursorHit(msg.x, msg.y))
412         status = Status::Pushed;
413       break;
414     case WM_LBUTTONUP:
415       if (status == Status::Pushed && CheckCursorHit(msg.x, msg.y))
416         OnClick();
417       break;
418     default:
419       break;
420     }
421   }
422
423   void Draw() {
424     switch (status) {
425     case Status::Idle:
426       putimage_alpha(region.left, region.top, &img_idle);
427       break;
428     case Status::Hovered:
429       putimage_alpha(region.left, region.top, &img_hovered);
430       break;
431     case Status::Pushed:
432       putimage_alpha(region.left, region.top, &img_pushed);
433       break;
434     }
```

```
435     }
436
437 protected:
438     virtual void OnClick() = 0;
439
440 private:
441     enum class Status {
442         Idle = 0,
443         Hovered,
444         Pushed
445     };
446
447 private:
448
449     //检测鼠标点击
450     bool CheckCursorHit(int x, int y) {
451         return x >= region.left && x <= region.right && y >= region.top && y <=
              region.bottom;
452     }
453
454 private:
455     RECT region;//描述位置和大小
456     IMAGE img_idle;
457     IMAGE img_hovered;
458     IMAGE img_pushed;
459     Status status = Status::Idle;
460 };
461
462
463 //开始游戏按钮
464 class StartGameButton :public Button {
465 public:
466     StartGameButton(RECT rect, LPCTSTR path_img_idle, LPCTSTR path_img_hovered,
          LPCTSTR path_img_pushed)
467         :Button(rect, path_img_idle, path_img_hovered, path_img_pushed) {}
468     ~StartGameButton() = default;
469
470 protected:
471     void OnClick() {
472         is_character_selection = true; // 进入角色选择界面
473     }
474 };
```

```
475
476 //退出游戏按钮
477 class QuitGameButton :public Button {
478 public:
479    QuitGameButton(RECT rect, LPCTSTR path_img_idle, LPCTSTR path_img_hovered,
        LPCTSTR path_img_pushed)
480       :Button(rect, path_img_idle, path_img_hovered, path_img_pushed) {}
481    ~QuitGameButton() = default;
482
483 protected:
484    void OnClick() {
485       running = false;
486    }
487 };
488
489 // 角色选择按钮
490 class CharacterButton :public Button {
491 public:
492    CharacterButton(RECT rect, LPCTSTR path_img_idle, LPCTSTR path_img_hovered,
        LPCTSTR path_img_pushed, int character_index)
493       :Button(rect, path_img_idle, path_img_hovered, path_img_pushed),
          character_index(character_index) {}
494    ~CharacterButton() = default;
495
496 protected:
497    void OnClick() {
498       is_game_started = true;
499       selected_character = character_index;
500       is_character_selection = false;
501       Sleep(50);
502       mciSendString(_T("play bgm repeat from 0"), NULL, 0, NULL);
503
504    }
505
506 private:
507    int character_index;
508    RECT region;
509 };
510
511 // 角色选择界面相关变量
512 static IMAGE img_character_bg;
513 static CharacterButton* btn_character[3] = { nullptr }; // 使用数组管理按钮
```

```
514 static bool char_selection_initialized = false;
515
516 // 初始化角色选择界面
517 void InitCharacterSelection() {
518     if (!char_selection_initialized) {
519         loadimage(&img_character_bg, _T("img/character_selection_bg.png"), WIDTH0,
                HEIGHT0);
520         int total_width = CHAR_BTN_WIDTH * 3 + CHAR_BTN_SPACING * 2;
521         int start_x = (WIDTH0 - total_width) / 2;
522
523         for (int i = 0; i < 3; i++) {
524             RECT rect = {
525                 start_x + i * (CHAR_BTN_WIDTH + CHAR_BTN_SPACING),
526                 CHAR_BTN_TOP,
527                 start_x + (i + 1) * CHAR_BTN_WIDTH + i * CHAR_BTN_SPACING,
528                 CHAR_BTN_TOP + CHAR_BTN_HEIGHT
529             };
530
531             TCHAR path_idle[64], path_hovered[64], path_pushed[64];
532             _stprintf_s(path_idle, _T("img/character%d_idle.png"), i);
533             _stprintf_s(path_hovered, _T("img/character%d_hovered.png"), i);
534             _stprintf_s(path_pushed, _T("img/character%d_pushed.png"), i);
535
536             btn_character[i] = new CharacterButton(rect, path_idle, path_hovered,
                    path_pushed, i);
537         }
538         char_selection_initialized = true;
539     }
540 }
541
542 // 绘制角色选择界面（仅负责绘制）
543 void DrawCharacterSelectionUI() {
544     putimage(0, 0, &img_character_bg);
545     setbkmode(TRANSPARENT);
546     settextcolor(RGB(80, 134, 85));
547     settextstyle(36, 0, _T("微软雅黑"));
548     outtextxy(WIDTH0 / 2 - 120, 180, _T("请选择你的宝可梦"));
549
550     for (int i = 0; i < 3; i++) {
551         btn_character[i]->Draw();
552     }
553 }
```

```
554
555 // 处理角色选择界面事件
556 void ProcessCharacterSelectionEvent(const ExMessage& msg) {
557     for (int i = 0; i < 3; i++) {
558         btn_character[i]->ProcessEvent(msg);
559     }
560 }
561
562 //生成新的敌人
563 void TryGenerateEnemy(std::vector<Enemy*>& enemy_list) {
564     const int INTERVAL = 100;
565     static int counter = 0;
566     if ((++counter) % INTERVAL == 0) {
567         enemy_list.push_back(new Enemy());
568     }
569 };
570
571 //更新子弹的位置
572 void UpdateBullets(std::vector<Bullet>& bullet_list, const Player& player) {
573     const double R_SPEED = 0.004;
574     const double T_SPEED = 0.004;
575     double radian_interval = 2 * 3.1415926 / bullet_list.size();
576     POINT player_position = player.GetPosition();
577     double radius = 100 + 25 * sin(GetTickCount() * R_SPEED);
578     for (size_t i = 0; i < bullet_list.size(); i++) {
579         double radian = GetTickCount() * T_SPEED + radian_interval * i;
580         bullet_list[i].position.x = player_position.x + player.PLAYER_WIDTH / 2 +
              (int)(radius * sin(radian));
581         bullet_list[i].position.y = player_position.y + player.PLAYER_HEIGHT / 2 +
              (int)(radius * cos(radian));
582     }
583 }
584
585 //绘制当前玩家得分
586 void DrawPlayerScore(int score) {
587     static TCHAR text[64];
588     _stprintf_s(text, _T("当前玩家得分为：%d"), score);
589     settextstyle(22, 0, _T("微软雅黑"));
590     setbkmode(TRANSPARENT);
591     settextcolor(RGB(80, 134, 85));
592     outtextxy(10, 10, text);
593 }
```

```
594
595
596 int main() {
597     initgraph(WIDTH0, HEIGHT0);
598
599     //加载三个角色的动画图集
600     atlas_player_left[0] = new Atlas(_T("img/character1_left_%d.png"), 6);
601     atlas_player_right[0] = new Atlas(_T("img/character1_right_%d.png"), 6);
602     atlas_player_left[1] = new Atlas(_T("img/character2_left_%d.png"), 6);
603     atlas_player_right[1] = new Atlas(_T("img/character2_right_%d.png"), 6);
604     atlas_player_left[2] = new Atlas(_T("img/character3_left_%d.png"), 6);
605     atlas_player_right[2] = new Atlas(_T("img/character3_right_%d.png"), 6);
606
607     atlas_enemy_left = new Atlas(_T("img/enemy_left_%d.png"), 6);
608     atlas_enemy_right = new Atlas(_T("img/enemy_right_%d.png"), 6);
609
610     mciSendString(_T("open mus/bgm.mp3 alias bgm"), NULL, 0, NULL);
611     mciSendString(_T("open mus/hit.wav alias hit"), NULL, 0, NULL);
612
613     int score = 0;
614     Player* player = nullptr;
615     ExMessage msg;
616     IMAGE img_menu;
617     IMAGE img_background;
618
619     bool is_move_up = false;
620     bool is_move_down = false;
621     bool is_move_left = false;
622     bool is_move_right = false;
623     std::vector<Enemy*>enemy_list;
624     std::vector<Bullet>bullet_list(3);
625
626     RECT region_btn_start_game, region_btn_quit_game;
627
628     region_btn_start_game.left = (WIDTH0 - BUTTON_WIDTH) / 2;
629     region_btn_start_game.right = region_btn_start_game.left + BUTTON_WIDTH;
630     region_btn_start_game.top = 430;
631     region_btn_start_game.bottom = region_btn_start_game.top + BUTTON_HEIGHT;
632
633     region_btn_quit_game.left = (WIDTH0 - BUTTON_WIDTH) / 2;
634     region_btn_quit_game.right = region_btn_quit_game.left + BUTTON_WIDTH;
635     region_btn_quit_game.top = 550;
```

```
636   region_btn_quit_game.bottom = region_btn_quit_game.top + BUTTON_HEIGHT;
637
638   StartGameButton btn_start_game = StartGameButton(region_btn_start_game,
639     _T("img/ui_start_idle.png"), _T("img/ui_start_hovered.png"),
          _T("img/ui_start_pushed.png"));
640   QuitGameButton btn_quit_game = QuitGameButton(region_btn_quit_game,
641     _T("img/ui_quit_idle.png"), _T("img/ui_quit_hovered.png"),
          _T("img/ui_quit_pushed.png"));
642
643
644   loadimage(&img_menu, _T("img/menu.png"), 1280, 720);
645   loadimage(&img_background, _T("img/background.png"), 1280, 720);
646
647   BeginBatchDraw();
648
649   while (running) {
650     DWORD begin_time = GetTickCount();
651     ExMessage msg;
652
653     // 统一消息处理
654     while (peekmessage(&msg)) {
655       if (is_game_started) {
656         if (player) player->ProcessEvent(msg);
657       }
658       else if (is_character_selection) {
659         ProcessCharacterSelectionEvent(msg); // 处理角色选择事件
660       }
661       else {
662         btn_start_game.ProcessEvent(msg);
663         btn_quit_game.ProcessEvent(msg);
664       }
665     }
666
667     // 游戏逻辑
668     if (is_game_started) {
669       if (!player) {
670         player = new Player();
671       }
672
673       player->Move();
674       UpdateBullets(bullet_list, *player);
675       TryGenerateEnemy(enemy_list);
```

```
676         //更新敌人位置
677         for (Enemy* enemy : enemy_list)
678             enemy->Move(*player);
679 
680         //检测子弹与敌人的碰撞
681         for (Enemy* enemy : enemy_list) {
682             for (const Bullet& bullet : bullet_list) {
683                 if (enemy->CheckBulletCollision(bullet)) {
684                     enemy->Hurt();
685                     score++;
686                 }
687             }
688         }
689 
690         //移除生命值归零的敌人
691         for (size_t i = 0; i < enemy_list.size(); i++) {
692             Enemy* enemy = enemy_list[i];
693             if (!enemy->CheckAlive()) {
694                 std::swap(enemy_list[i], enemy_list.back());
695                 enemy_list.pop_back();
696                 delete enemy;
697             }
698         }
699 
700         // 检测敌人与玩家的碰撞
701         bool is_game_over = false; // 增加游戏结束标志
702         for (Enemy* enemy : enemy_list) {
703             if (enemy->CheckPlayerCollision(*player)) {
704                 is_game_over = true;
705                 break; // 发现碰撞立即跳出循环
706             }
707         }
708 
709         if (is_game_over) {
710             // 游戏结束逻辑
711             TCHAR text[128];
712             _stprintf_s(text, _T("最终得分为: %d! \n点击确定返回主菜单"), score);
713 
714             // 停止音乐
715             mciSendString(_T("stop bgm"), NULL, 0, NULL);
716 
717             // 显示结果弹窗
```

```
718             if (MessageBox(GetHWnd(), text, _T("游戏结束"), MB_OKCANCEL |
                    MB_ICONINFORMATION) == IDOK) {
719                 // 重置游戏状态
720                 is_game_started = false;
721                 is_character_selection = false;
722                 score = 0;
723                 player->position = { 300, 300 }; // 重置玩家位置
724
725                 //删除玩家对象
726                 if (player) {
727                     delete player;
728                     player = nullptr;
729                 }
730                 // 清空敌人
731                 for (auto& enemy : enemy_list) delete enemy;
732                 enemy_list.clear();
733             }
734             else {
735                 running = false; // 如果点取消则退出游戏
736             }
737         }
738     }
739
740     cleardevice();
741     //绘图
742
743     if (is_game_started) {
744         putimage(0, 0, &img_background);//铺背景图片
745
746         player->Draw(1000 / 144);
747         for (Enemy* enemy : enemy_list)
748             enemy->Draw(1000 / 144);
749         for (const Bullet& bullet : bullet_list)
750             bullet.Draw();
751         DrawPlayerScore(score);
752     }
753     else if (is_character_selection) {
754         InitCharacterSelection(); // 确保初始化
755         DrawCharacterSelectionUI(); // 绘制角色选择界面
756     }
757     else {
758         putimage(0, 0, &img_menu);
```

```
759         btn_start_game.Draw();
760         btn_quit_game.Draw();
761       }
762
763       FlushBatchDraw();
764
765       DWORD end_time = GetTickCount();
766       DWORD delta_time = end_time - begin_time;
767       if (delta_time < 1000 / 60) {
768         Sleep(1000 / 60 - delta_time);
769       }
770     }
771
772     // 释放所有角色图集资源
773     for (int i = 0; i < 3; i++) {
774       delete atlas_player_left[i];
775       delete atlas_player_right[i];
776     }
777     delete atlas_enemy_left;
778     delete atlas_enemy_right;
779
780     // 释放角色选择按钮资源
781     for (int i = 0; i < 3; i++) {
782       delete btn_character[i];
783     }
784
785     EndBatchDraw();
786
787     while (!enemy_list.empty()) {
788       delete enemy_list.back();
789       enemy_list.pop_back();
790     }
791
792     return 0;
793 }
```