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 13 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

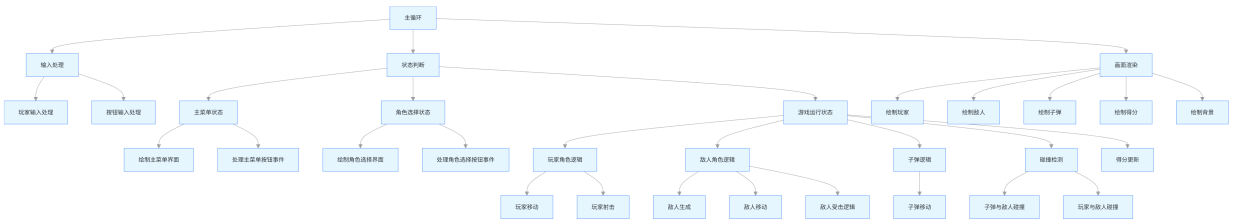

图 5.1: 整体架构实现思路

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：

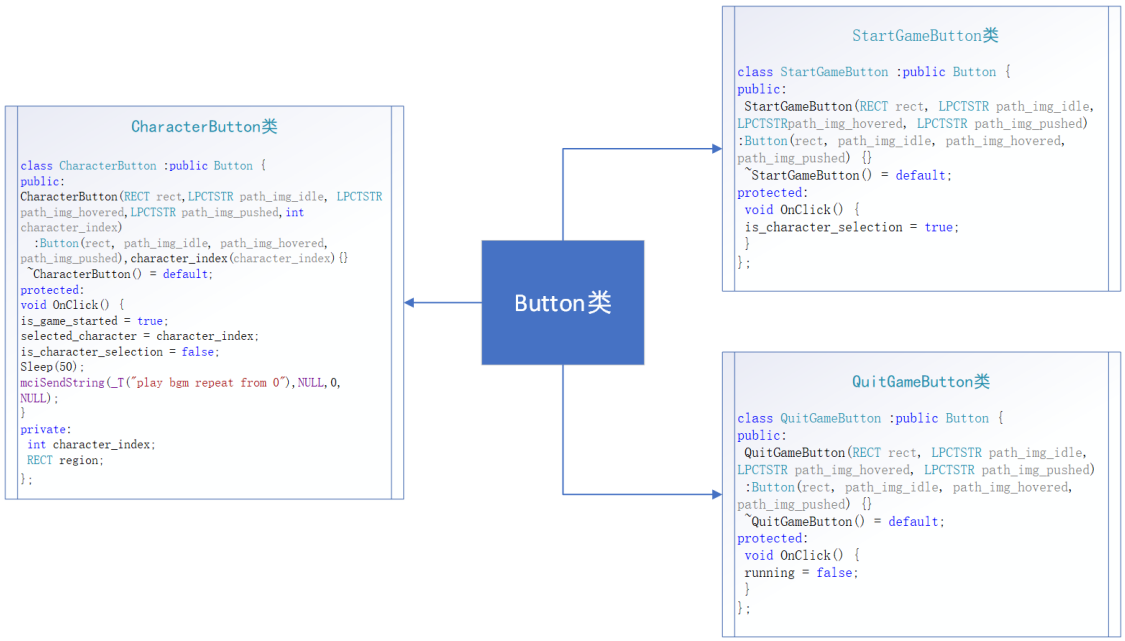

图 5.2: button 类图

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

这里的主要思想是将环绕在玩家周围的子弹等效为一个点，看其是否在敌人的矩形中；把敌人也等效为点，来判断其是否在玩家矩形内。

```
bool CheckBulletCollision(const Bullet& bullet) {
 bool is_overlap_x = bullet.position.x >= position.x && bullet.position.x <=
    position.x + FRAME_WIDTH;
 bool is_overlap_y = bullet.position.y >= position.y && bullet.position.y <=
    position.y + FRAME_HEIGHT;
```

```
 return is_overlap_x && is_overlap_y;
}

bool CheckPlayerCollision(const Player& player) {
 POINT check_position = { position.x + FRAME_WIDTH / 2,position.y + FRAME_HEIGHT /
     2 };
 bool is_overlap_x = player.position.x >= position.x && player.position.x
     <=position.x + FRAME_WIDTH;
 bool is_overlap_y = player.position.y >= position.y && player.position.y
     <=position.y + FRAME_HEIGHT;
 return is_overlap_x && is_overlap_y;
}
```

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

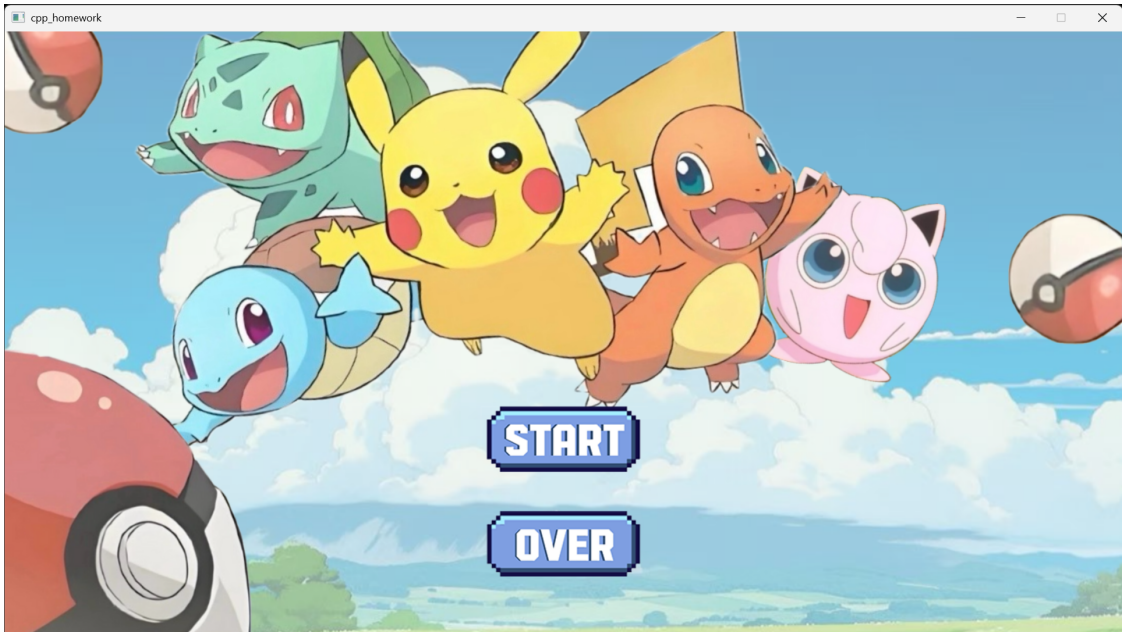

图 6.3: 主菜单

6.2 角色选择页面

图 6.4: 角色选择界面

6.3 游戏页面

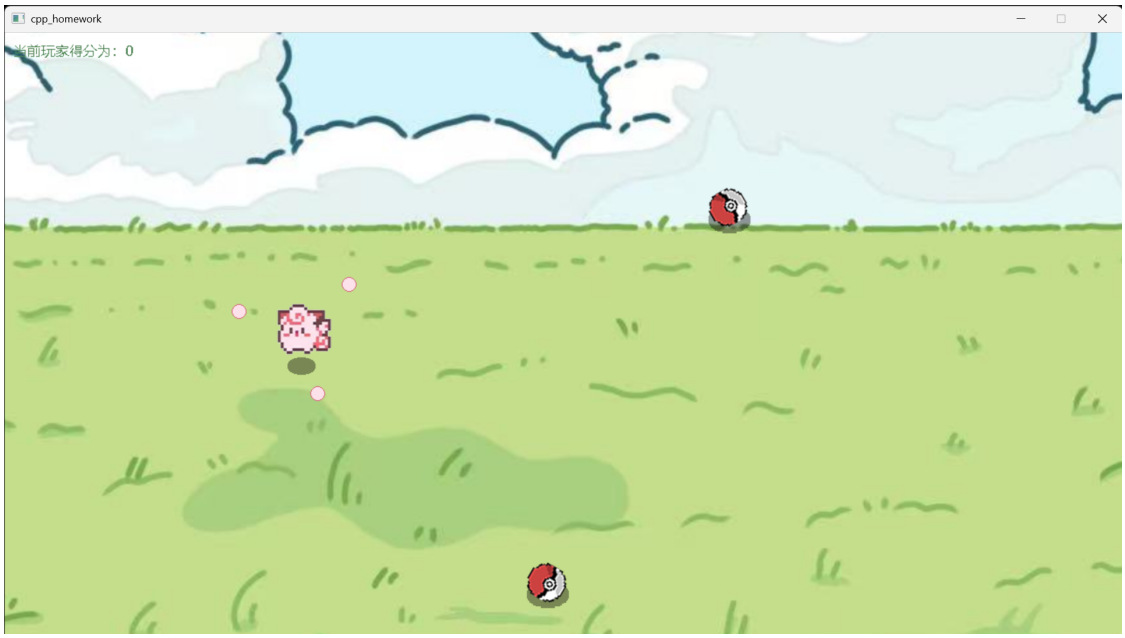

图 6.5: 游戏界面

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

// 角色选择按钮布局参数
const int CHAR_BTN_WIDTH = 160; // 单个按钮宽度
const int CHAR_BTN_HEIGHT = 200; // 单个按钮高度
const int CHAR_BTN_SPACING = 40; // 按钮间距
const int CHAR_BTN_TOP = 300;   // 距离顶部的固定位置

//绘制透明底图片的自定义函数
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 });
}

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);//利用pushback函数把图片对象的指针添加到容器里面
    }
```

```
    }
    ~Atlas() {
        for (size_t i = 0; i < frame_list.size(); i++) {
            delete frame_list[i];
        }
    }

public:
    std::vector<IMAGE*>frame_list;
};

// 存储不同角色的动画图集
Atlas* atlas_player_left[3]; // 3个角色的左移动画
Atlas* atlas_player_right[3]; // 3个角色的右移动画
Atlas* atlas_enemy_left;
Atlas* atlas_enemy_right;

class Animation {
public:
    Animation(Atlas* atlas, int interval) {
        anim_atlas = atlas;
        interval_ms = interval;
    }

    ~Animation() = default;

    //动画播放
    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;//动画计时器
    int idx_frame = 0;//动画帧索引
    int interval_ms = 0;
```

```
83    Atlas* anim_atlas;
84 };
85
86 const int WIDTH0 = 1280;//页面宽度
87 const int HEIGHT0 = 720;//页面高度
88 const int BUTTON_WIDTH = 190;
89 const int BUTTON_HEIGHT = 75;
90
91 class Player {
92 public:
93    const int PLAYER_WIDTH = 80;//玩家宽度
94    const int PLAYER_HEIGHT = 80;//玩家高度
95    POINT position = { 300,300 };//初始化玩家位置
96 public:
97    Player( ) {
98       loadimage(&img_shadow, _T("img/shadow_player.png"));
99       // 使用选中的角色图集
100      anim_left = new Animation(atlas_player_left[selected_character], 45);
101      anim_right = new Animation(atlas_player_right[selected_character], 45);
102   };
103
104   ~Player() {
105      delete anim_left;
106      delete anim_right;
107   }
108
109   void ProcessEvent(const ExMessage& msg) {
110      //通过按键来实现角色移动
111      if (msg.message == WM_KEYDOWN) {
112         switch (msg.vkcode) {
113         case VK_UP:
114            is_move_up = true;
115            break;
116         case VK_DOWN:
117            is_move_down = true;
118            break;
119         case VK_LEFT:
120            is_move_left = true;
121            break;
122         case VK_RIGHT:
123            is_move_right = true;
124            break;
```

```
125         }
126     }
127     else if (msg.message == WM_KEYUP) {
128         switch (msg.vkcode) {
129         case VK_UP:
130             is_move_up = false;
131             break;
132         case VK_DOWN:
133             is_move_down = false;
134             break;
135         case VK_LEFT:
136             is_move_left = false;
137             break;
138         case VK_RIGHT:
139             is_move_right = false;
140             break;
141         }
142     }
143 }
144 
145 void Move() {
146     //利用向量知识来解决斜方向移动速度特别快的问题
147     int dir_x = is_move_right - is_move_left;
148     int dir_y = is_move_down - is_move_up;;
149     double len_dir = sqrt(dir_x * dir_x + dir_y * dir_y);
150     if (len_dir != 0) {
151         double normalized_x = dir_x / len_dir;
152         double normalized_y = dir_y / len_dir;
153         position.x += (int)(SPEED * normalized_x);
154         position.y += (int)(SPEED * normalized_y);
155     }
156 
157     //对玩家位置进行校准，防止玩家超出游戏界面
158     if (position.x < 0)
159         position.x = 0;
160     if (position.y < 0)
161         position.y = 0;
162     if (position.x + PLAYER_WIDTH > WIDTH0)
163         position.x = WIDTH0 - PLAYER_WIDTH;
164     if (position.y + PLAYER_HEIGHT > HEIGHT0)
165         position.y = HEIGHT0 - PLAYER_HEIGHT;
166 }
```

```
167
168     void Draw(int delta) {
169         int pos_shadow_x = position.x + (PLAYER_WIDTH / 2 - SHADOW_WIDTH / 2);
170         int pos_shadow_y = position.y + PLAYER_HEIGHT - 8;//偏移一点点
171         putimage_alpha(pos_shadow_x, pos_shadow_y, &img_shadow);
172
173         static bool facing_left = false;
174         int dir_x = is_move_right - is_move_left;
175         if (dir_x < 0)
176             facing_left = true;
177         else if (dir_x > 0)
178             facing_left = false;
179
180         if (facing_left)
181             anim_left->Play(position.x, position.y, delta);
182         else
183             anim_right->Play(position.x, position.y, delta);
184     }
185
186     const POINT& GetPosition() const {
187         return position;
188     };
189
190 private:
191     const int SHADOW_WIDTH = 32;//影子大小
192     const int SPEED = 5;//定义速度
193
194 private:
195     IMAGE img_shadow;
196     Animation* anim_left;
197     Animation* anim_right;
198     bool is_move_up = false;
199     bool is_move_down = false;
200     bool is_move_left = false;
201     bool is_move_right = false;
202 };
203
204 class Bullet {
205 public:
206     POINT position = { 0,0 };
207
208 public:
```

```
209     Bullet() = default;
210     ~Bullet() = default;
211     void Draw() const {
212         // 根据选择的角色改变子弹颜色
213         switch (selected_character) {
214         case 0:
215             setfillcolor(RGB(252, 225, 235));
216             setlinecolor(RGB(215, 108, 124));
217             break;
218         case 1:
219             setlinecolor(RGB(134, 218, 227));
220             setfillcolor(RGB(51, 166, 223));
221             break;
222         case 2:
223             setlinecolor(RGB(250, 210, 89));
224             setfillcolor(RGB(255, 183, 43));
225             break;
226         default:
227             setlinecolor(RED);
228             setfillcolor(RED);
229         }
230         fillcircle(position.x, position.y, R);
231     }
232     int GetR()const {
233         return R;
234     }
235 private:
236     const int R = 8;
237 };
238
239
240 class Enemy {
241 public:
242     //敌人生成边界
243     enum class SpawnEdge {
244         Up = 0,
245         Down,
246         Left,
247         Right
248     };
249     Enemy() {
250         loadimage(&img_shadow, _T("img/shadow_enemy.png"));
```

```
251     anim_left = new Animation(atlas_enemy_left, 45);
252     anim_right = new Animation(atlas_enemy_right, 45);
253
254     SpawnEdge edge = (SpawnEdge)(rand() % 4);
255     switch (edge) {
256     case SpawnEdge::Up:
257       position.x = rand() % WIDTH0;
258       position.y = -ENEMY_HEIGHT;
259       break;
260     case SpawnEdge::Down:
261       position.x = rand() % WIDTH0;
262       position.y = HEIGHT0;
263       break;
264     case SpawnEdge::Left:
265       position.x = -ENEMY_WIDTH;
266       position.y = rand() % HEIGHT0;
267       break;
268     case SpawnEdge::Right:
269       position.x = WIDTH0;
270       position.y = rand() % HEIGHT0;
271       break;
272     default:
273       break;
274     }
275   }
276   bool CheckBulletCollision(const Bullet& bullet) {
277     //将子弹等效为点，判断是否在敌人矩形内
278     bool is_overlap_x = bullet.position.x >= position.x && bullet.position.x <=
          position.x + ENEMY_WIDTH;
279     bool is_overlap_y = bullet.position.y >= position.y && bullet.position.y <=
          position.y + ENEMY_HEIGHT;
280     return is_overlap_x && is_overlap_y;
281   }
282   bool CheckPlayerCollision(const Player& player) {
283     //将敌人中心位置等效为点，判断是否在玩家矩形内
284     POINT check_position = { position.x + ENEMY_WIDTH / 2,position.y +
          ENEMY_HEIGHT / 2 };
285     bool is_overlap_x = player.position.x >= position.x && player.position.x <=
          position.x + ENEMY_WIDTH;
286     bool is_overlap_y = player.position.y >= position.y && player.position.y <=
          position.y + ENEMY_HEIGHT;
287     return is_overlap_x && is_overlap_y;
```

```
288   }
289   void Move(const Player& player) {
290      const POINT& player_position = player.GetPosition();
291      int dir_x = player_position.x - position.x;
292      facing_left = (dir_x < 0);
293      int dir_y = player_position.y - position.y;
294      double len_dir = sqrt(dir_x * dir_x + dir_y * dir_y);
295      if (len_dir != 0) {
296         double normalized_x = dir_x / len_dir;
297         double normalized_y = dir_y / len_dir;
298         position.x += (int)(SPEED * normalized_x);
299         position.y += (int)(SPEED * normalized_y);
300      }
301   }
302
303   void Draw(int delta) {
304      int pos_shadow_x = position.x + (ENEMY_WIDTH / 2 - SHADOW_WIDTH / 2);
305      int pos_shadow_y = position.y + ENEMY_HEIGHT - 31;//偏移一点点
306      putimage_alpha(pos_shadow_x, pos_shadow_y, &img_shadow);
307      if (facing_left)
308         anim_left->Play(position.x, position.y, delta);
309      else
310         anim_right->Play(position.x, position.y, delta);
311   }
312
313   const POINT& GetPosition() const {
314      return position;
315   }
316
317   ~Enemy() {
318      delete anim_left;
319      delete anim_right;
320   }
321
322   void Hurt() {
323      alive = false;
324   }
325
326   bool CheckAlive() {
327      return alive;
328   }
329
```

```
330 private:
331     const int ENEMY_WIDTH = 60;//敌人宽度
332     const int ENEMY_HEIGHT = 60;//敌人高度
333     const int SHADOW_WIDTH = 48;//影子大小
334     const int SPEED = 4;//定义速度
335 
336 private:
337     IMAGE img_shadow;
338     Animation* anim_left;
339     Animation* anim_right;
340     POINT position = { 0,0 };//初始化敌人位置;
341     bool facing_left = false;
342     bool alive = true;
343     bool is_move_up = false;
344     bool is_move_down = false;
345     bool is_move_left = false;
346     bool is_move_right = false;
347 };
348 
349 //按钮类
350 class Button {
351 public:
352     Button(RECT rect, LPCTSTR path_img_idle, LPCTSTR path_img_hovered, LPCTSTR
          path_img_pushed) {
353         region = rect;
354         loadimage(&img_idle, path_img_idle);
355         loadimage(&img_hovered, path_img_hovered);
356         loadimage(&img_pushed, path_img_pushed);
357     }
358 
359     ~Button() = default;
360 
361     void ProcessEvent(const ExMessage& msg) {
362         switch (msg.message) {
363         case WM_MOUSEMOVE:
364             if (status == Status::Idle && CheckCursorHit(msg.x, msg.y))
365                 status = Status::Hovered;
366             else if (status == Status::Hovered && !CheckCursorHit(msg.x, msg.y))
367                 status = Status::Idle;
368             break;
369         case WM_LBUTTONDOWN:
370             if (CheckCursorHit(msg.x, msg.y))
```

```
371             status = Status::Pushed;
372         break;
373       case WM_LBUTTONUP:
374         if (status == Status::Pushed && CheckCursorHit(msg.x, msg.y))
375           OnClick();
376         break;
377       default:
378         break;
379       }
380     }
381
382     void Draw() {
383       switch (status) {
384       case Status::Idle:
385         putimage_alpha(region.left, region.top, &img_idle);
386         break;
387       case Status::Hovered:
388         putimage_alpha(region.left, region.top, &img_hovered);
389         break;
390       case Status::Pushed:
391         putimage_alpha(region.left, region.top, &img_pushed);
392         break;
393       }
394     }
395
396 protected:
397     virtual void OnClick() = 0;
398 private:
399     enum class Status {
400       Idle = 0,
401       Hovered,
402       Pushed
403     };
404
405 private:
406
407     //检测鼠标点击
408     bool CheckCursorHit(int x, int y) {
409       return x >= region.left && x <= region.right && y >= region.top && y <=
          region.bottom;
410     }
411
```

```
412 private:
413     RECT region;//描述位置和大小
414     IMAGE img_idle;
415     IMAGE img_hovered;
416     IMAGE img_pushed;
417     Status status = Status::Idle;
418 };
419
420 //开始游戏按钮
421 class StartGameButton :public Button {
422 public:
423     StartGameButton(RECT rect, LPCTSTR path_img_idle, LPCTSTR path_img_hovered,
        LPCTSTR path_img_pushed)
424         :Button(rect, path_img_idle, path_img_hovered, path_img_pushed) {}
425     ~StartGameButton() = default;
426
427 protected:
428     void OnClick() {
429         is_character_selection = true; // 进入角色选择界面
430     }
431 };
432
433 //退出游戏按钮
434 class QuitGameButton :public Button {
435 public:
436     QuitGameButton(RECT rect, LPCTSTR path_img_idle, LPCTSTR path_img_hovered,
        LPCTSTR path_img_pushed)
437         :Button(rect, path_img_idle, path_img_hovered, path_img_pushed) {}
438     ~QuitGameButton() = default;
439
440 protected:
441     void OnClick() {
442         running = false;
443     }
444 };
445
446 // 角色选择按钮
447 class CharacterButton :public Button {
448 public:
449     CharacterButton(RECT rect, LPCTSTR path_img_idle, LPCTSTR path_img_hovered,
        LPCTSTR path_img_pushed, int character_index)
450         :Button(rect, path_img_idle, path_img_hovered, path_img_pushed),
```

```
        character_index(character_index) {}
    ~CharacterButton() = default;
protected:
    void OnClick() {
        is_game_started = true;
        selected_character = character_index;
        is_character_selection = false;
        Sleep(50);
        mciSendString(_T("play bgm repeat from 0"), NULL, 0, NULL);

    }
private:
    int character_index;
    RECT region;
};

//生成新的敌人
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());
    }
};

//更新子弹的位置
void UpdateBullets(std::vector<Bullet>& bullet_list, const Player& player) {
    const double R_SPEED = 0.004;
    const double T_SPEED = 0.004;
    double radian_interval = 2 * 3.1415926 / bullet_list.size();
    POINT player_position = player.GetPosition();
    double radius = 100 + 25 * sin(GetTickCount() * R_SPEED);
    for (size_t i = 0; i < bullet_list.size(); i++) {
        double radian = GetTickCount() * T_SPEED + radian_interval * i;
        bullet_list[i].position.x = player_position.x + player.PLAYER_WIDTH / 2 +
            (int)(radius * sin(radian));
        bullet_list[i].position.y = player_position.y + player.PLAYER_HEIGHT / 2 +
            (int)(radius * cos(radian));
    }
}

//绘制当前玩家得分
```

```
490 void DrawPlayerScore(int score) {
491     static TCHAR text[64];
492     _stprintf_s(text, _T("当前玩家得分为：%d"), score);
493     settextstyle(22, 0, _T("微软雅黑"));
494     setbkmode(TRANSPARENT);
495     settextcolor(RGB(80, 134, 85));
496     outtextxy(10, 10, text);
497 }
498
499 // 绘制角色选择界面
500 void DrawCharacterSelection() {
501     static bool is_init = false;
502     static IMAGE img_character_bg;
503     static CharacterButton* btn_character[3] = { nullptr }; // 使用数组管理按钮
504
505     // 首次加载时初始化
506     if (!btn_character[0]&&!is_init) {
507         loadimage(&img_character_bg, _T("img/character_selection_bg.png"), WIDTH0,
              HEIGHT0);
508
509         // 计算起始X坐标（整体居中）
510         int total_width = CHAR_BTN_WIDTH * 3 + CHAR_BTN_SPACING * 2;
511         int start_x = (WIDTH0 - total_width) / 2;
512
513         // 初始化三个按钮
514         for (int i = 0; i < 3; i++) {
515             RECT rect = {
516                 start_x + i * (CHAR_BTN_WIDTH + CHAR_BTN_SPACING),
517                 CHAR_BTN_TOP,
518                 start_x + (i + 1) * CHAR_BTN_WIDTH + i * CHAR_BTN_SPACING,
519                 CHAR_BTN_TOP + CHAR_BTN_HEIGHT
520             };
521
522             TCHAR path_idle[64], path_hovered[64], path_pushed[64];
523             _stprintf_s(path_idle, _T("img/character%d_idle.png"), i);
524             _stprintf_s(path_hovered, _T("img/character%d_hovered.png"), i);
525             _stprintf_s(path_pushed, _T("img/character%d_pushed.png"), i);
526
527             btn_character[i] = new CharacterButton(rect, path_idle, path_hovered,
                  path_pushed, i);
528         }
529         is_init = true;
```

```
530     }
531
532     // 绘制背景
533     putimage(0, 0, &img_character_bg);
534
535     // 绘制标题
536     setbkmode(TRANSPARENT);
537     settextcolor(RGB(80, 134, 85));
538     settextstyle(36, 0, _T("微软雅黑"));
539     outtextxy(WIDTH0 / 2 - 120, 180, _T("请选择你的宝可梦"));
540
541     // 处理事件和绘制
542     ExMessage msg;
543     while (peekmessage(&msg)) {
544
545         for (int i = 0; i < 3; i++) {
546             btn_character[i]->ProcessEvent(msg);
547         }
548     }
549
550     for (int i = 0; i < 3; i++) {
551         btn_character[i]->Draw();
552     }
553 }
554
555 int main() {
556     initgraph(WIDTH0, HEIGHT0);
557     //加载三个角色的动画图集
558     atlas_player_left[0] = new Atlas(_T("img/character1_left_%d.png"), 6);
559     atlas_player_right[0] = new Atlas(_T("img/character1_right_%d.png"), 6);
560     atlas_player_left[1] = new Atlas(_T("img/character2_left_%d.png"), 6);
561     atlas_player_right[1] = new Atlas(_T("img/character2_right_%d.png"), 6);
562     atlas_player_left[2] = new Atlas(_T("img/character3_left_%d.png"), 6);
563     atlas_player_right[2] = new Atlas(_T("img/character3_right_%d.png"), 6);
564
565     atlas_enemy_left = new Atlas(_T("img/enemy_left_%d.png"), 6);
566     atlas_enemy_right = new Atlas(_T("img/enemy_right_%d.png"), 6);
567
568     mciSendString(_T("open mus/bgm.mp3 alias bgm"), NULL, 0, NULL);
569     mciSendString(_T("open mus/hit.wav alias hit"), NULL, 0, NULL);
570
571     int score = 0;
```

```
572   Player* player = nullptr;
573   ExMessage msg;
574   IMAGE img_menu;
575   IMAGE img_background;
576
577   bool is_move_up = false;
578   bool is_move_down = false;
579   bool is_move_left = false;
580   bool is_move_right = false;
581   std::vector<Enemy*>enemy_list;
582   std::vector<Bullet>bullet_list(3);
583
584   RECT region_btn_start_game, region_btn_quit_game;
585
586   region_btn_start_game.left = (WIDTH0 - BUTTON_WIDTH) / 2;
587   region_btn_start_game.right = region_btn_start_game.left + BUTTON_WIDTH;
588   region_btn_start_game.top = 430;
589   region_btn_start_game.bottom = region_btn_start_game.top + BUTTON_HEIGHT;
590
591   region_btn_quit_game.left = (WIDTH0 - BUTTON_WIDTH) / 2;
592   region_btn_quit_game.right = region_btn_quit_game.left + BUTTON_WIDTH;
593   region_btn_quit_game.top = 550;
594   region_btn_quit_game.bottom = region_btn_quit_game.top + BUTTON_HEIGHT;
595
596   StartGameButton btn_start_game = StartGameButton(region_btn_start_game,
597      _T("img/ui_start_idle.png"), _T("img/ui_start_hovered.png"),
          _T("img/ui_start_pushed.png"));
598   QuitGameButton btn_quit_game = QuitGameButton(region_btn_quit_game,
599      _T("img/ui_quit_idle.png"), _T("img/ui_quit_hovered.png"),
          _T("img/ui_quit_pushed.png"));
600
601   loadimage(&img_menu, _T("img/menu.png"), 1280, 720);
602   loadimage(&img_background, _T("img/background.png"), 1280, 720);
603
604   BeginBatchDraw();
605
606   while (running) {
607      DWORD begin_time = GetTickCount();
608
609      while (peekmessage(&msg)) {
610         if (is_game_started) {
611            player->ProcessEvent(msg);
```

```
612         }
613         else if (is_character_selection) {
614             // 角色选择界面的消息处理在DrawCharacterSelection函数中完成
615             DrawCharacterSelection();
616         }
617         else {
618             btn_start_game.ProcessEvent(msg);
619             btn_quit_game.ProcessEvent(msg);
620         }
621     }
622 
623     if (is_game_started) {
624         if (!player) {
625             player = new Player();
626         }
627 
628         player->Move();
629         UpdateBullets(bullet_list, *player);
630         TryGenerateEnemy(enemy_list);
631         //更新敌人位置
632         for (Enemy* enemy : enemy_list)
633             enemy->Move(*player);
634 
635         //检测子弹与敌人的碰撞
636         for (Enemy* enemy : enemy_list) {
637             for (const Bullet& bullet : bullet_list) {
638                 if (enemy->CheckBulletCollision(bullet)) {
639                     enemy->Hurt();
640                     score++;
641                 }
642             }
643         }
644 
645         //移除生命值归零的敌人
646         for (size_t i = 0; i < enemy_list.size(); i++) {
647             Enemy* enemy = enemy_list[i];
648             if (!enemy->CheckAlive()) {
649                 std::swap(enemy_list[i], enemy_list.back());
650                 enemy_list.pop_back();
651                 delete enemy;
652             }
653         }
```

```
654
655         // 检测敌人与玩家的碰撞
656         bool is_game_over = false; // 增加游戏结束标志
657         for (Enemy* enemy : enemy_list) {
658             if (enemy->CheckPlayerCollision(*player)) {
659                 is_game_over = true;
660                 break; // 发现碰撞立即跳出循环
661             }
662         }
663
664         if (is_game_over) {
665             // 游戏结束逻辑
666             TCHAR text[128];
667             _stprintf_s(text, _T("最终得分为: %d! \n点击确定返回主菜单"), score);
668
669             // 停止音乐
670             mciSendString(_T("stop bgm"), NULL, 0, NULL);
671
672             // 显示结果弹窗
673             if (MessageBox(GetHWnd(), text, _T("游戏结束"), MB_OKCANCEL |
                    MB_ICONINFORMATION) == IDOK) {
674                 // 重置游戏状态
675                 is_game_started = false;
676                 is_character_selection = false;
677                 score = 0;
678                 player->position = { 300, 300 }; // 重置玩家位置
679
680                 //删除玩家对象
681                 if (player) {
682                     delete player;
683                     player = nullptr;
684                 }
685                 // 清空敌人
686                 for (auto& enemy : enemy_list) delete enemy;
687                 enemy_list.clear();
688             }
689             else {
690                 running = false; // 如果点取消则退出游戏
691             }
692         }
693     }
694
```

```
695     cleardevice();
696     //绘图
697     if (is_game_started) {
698         putimage(0, 0, &img_background);//铺背景图片
699 
700         player->Draw(1000 / 144);
701         for (Enemy* enemy : enemy_list)
702             enemy->Draw(1000 / 144);
703         for (const Bullet& bullet : bullet_list)
704             bullet.Draw();
705         DrawPlayerScore(score);
706     }
707     else if (is_character_selection) {
708         DrawCharacterSelection(); // 绘制角色选择界面
709     }
710     else {
711         putimage(0, 0, &img_menu);
712         btn_start_game.Draw();
713         btn_quit_game.Draw();
714     }
715 
716     FlushBatchDraw();
717 
718     DWORD end_time = GetTickCount();
719     DWORD delta_time = end_time - begin_time;
720     if (delta_time < 1000 / 60) {
721         Sleep(1000 / 60 - delta_time);
722     }
723 }
724 
725 // 释放所有角色图集资源
726 for (int i = 0; i < 3; i++) {
727     delete atlas_player_left[i];
728     delete atlas_player_right[i];
729 }
730 delete atlas_enemy_left;
731 delete atlas_enemy_right;
732 
733 EndBatchDraw();
734 
735 while (!enemy_list.empty()) {
736     delete enemy_list.back();
```

```
737         enemy_list.pop_back();
738     }
739 
740     return 0;
741 }
```