南開大學

《高级语言程序设计 2-2》实验报告

基于 Raylib 的五子棋小游戏

姓名：李懋
学号：2213189
学院：软件学院

2025-05-15

C++大作业_五子棋

文档目录:

一、作业题目

基于Raylib的五子棋小游戏

二、开发环境

- 操作系统: macOS Sequoia 15.0

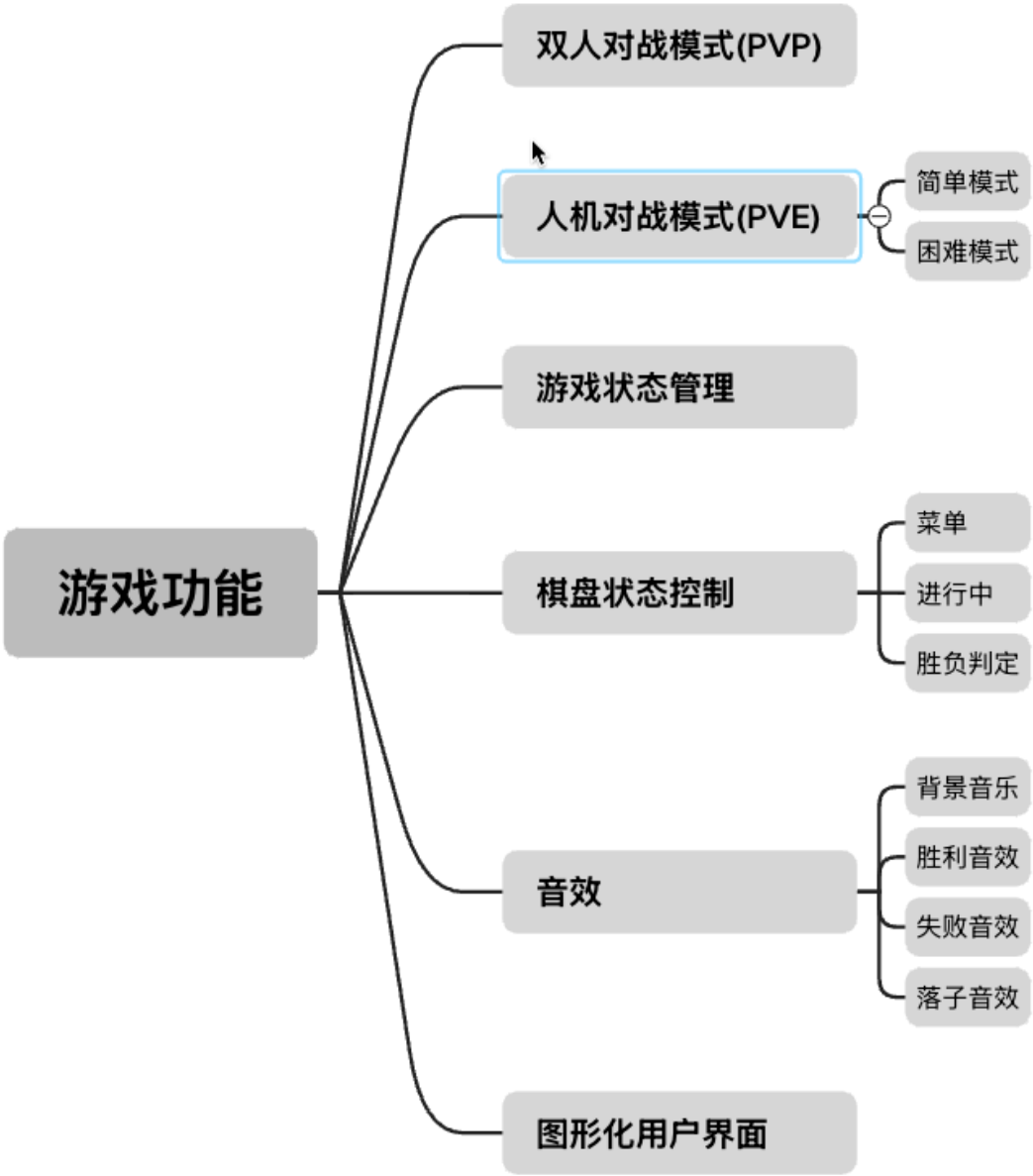
- IDE: VsCode
- 图形化工具: Raylib

三、项目概述

项目功能

本项目是一个基于Raylib库开发的五子棋游戏，支持以下功能：

- 双人对战模式(PVP)
- 人机对战模式(PVE)
  - 简单
  - 困难
- 游戏状态管理
  - 菜单
  - 进行中
  - 胜负判定
- 棋盘状态控制
  - 清空棋盘
  - 悔棋
- 游戏音效
  - 背景音乐
  - 胜利音效
  - 失败音效
  - 落子音效
- 图形化用户界面

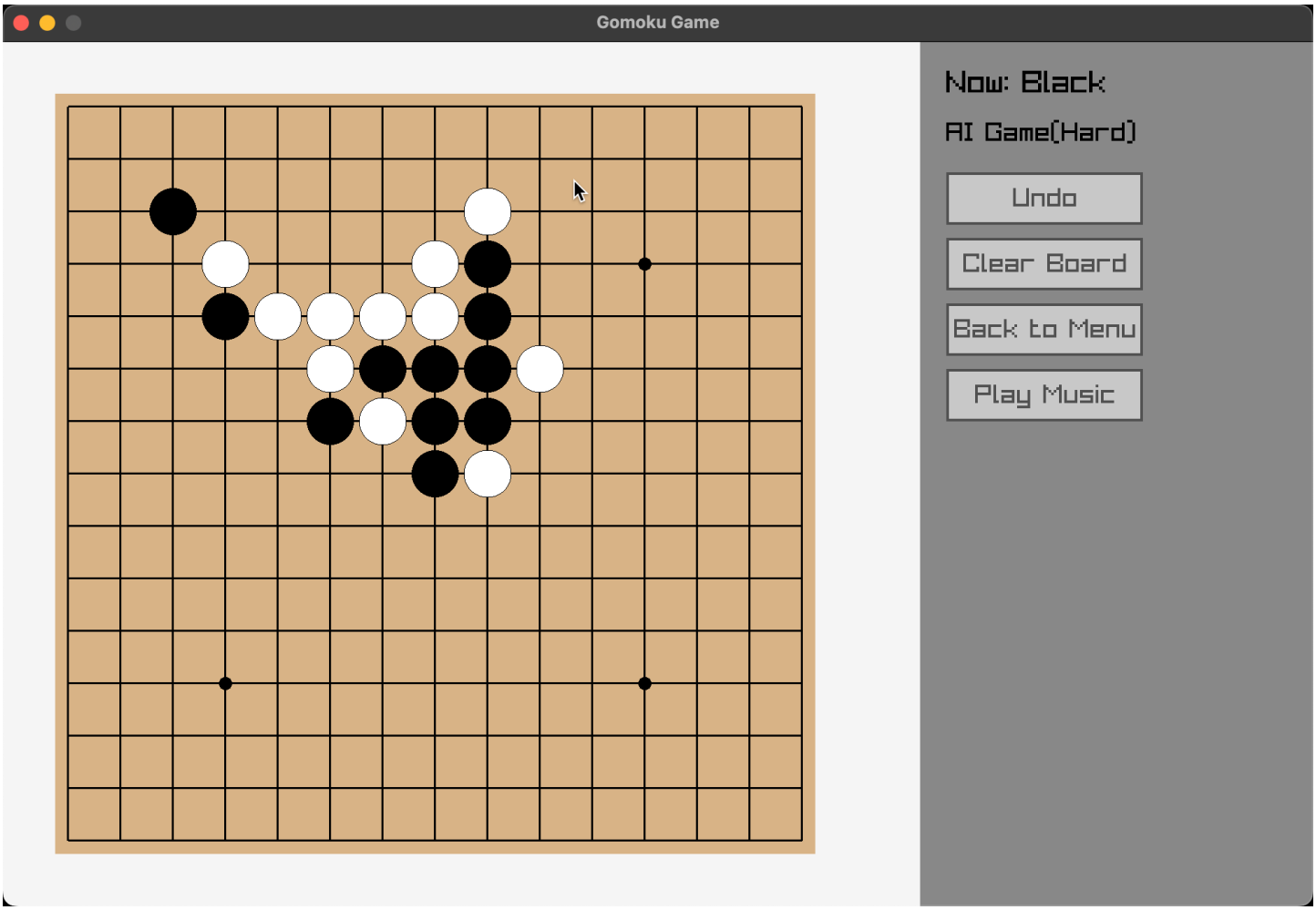

游戏功能
双人对战模式(PVP)
人机对战模式(PVE)
简单模式
困难模式
游戏状态管理
棋盘状态控制
菜单
进行中
胜负判定
音效
背景音乐
胜利音效
失败音效
落子音效
图形化用户界面

效果预览

主页菜单:


Gomoku Game
GOMOKU
Pair Game
AI Game[Easy]
AI Game[Hard]

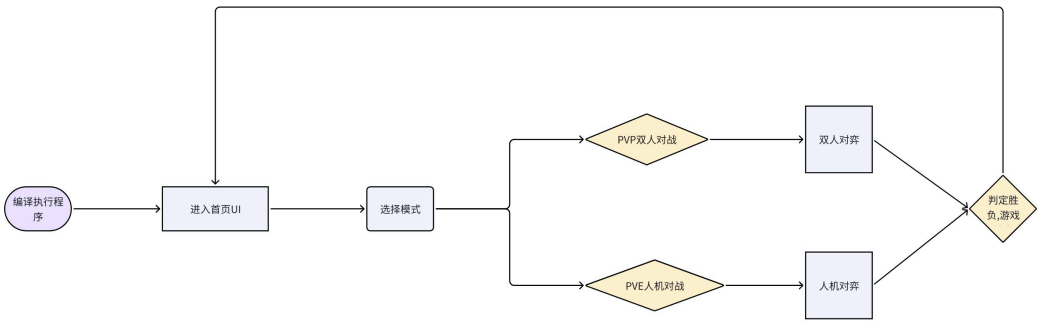
游戏界面:


Gomoku Game
Now: Black
AI Game[Hard]
Undo
Clear Board
Back to Menu
Play Music

四、课题要求

1. 面向对象设计
2. 实现完整游戏功能
3. 包含AI对战功能
4. 美观的用户UI

五、主要流程



六、类图

枚举类

代码块

```
+--------------------+
|     enum Piece     |
+--------------------+
| PIECE_EMPTY = 0    |
| PIECE_BLACK = 1    |
```

```
| PIECE_WHITE = 2   |
+-------------------+

+----------------------+
|   enum GameState     |
+----------------------+
| STATE_MENU           |
| STATE_PLAYING        |
| STATE_BLACK_WIN      |
| STATE_WHITE_WIN      |
+----------------------+

+----------------------+
|    enum GameMode     |
+----------------------+
| MODE_PVP             |
| MODE_PVE_EASY        |
| MODE_PVE_HARD        |
+----------------------+
```

结构体Button

代码块

```
+---------------------------------+
|           struct Button         |
+---------------------------------+
| - bounds: Rectangle             |
| - text: const char*             |
| - isHovered: bool               |
+---------------------------------+
| + CreateButton()                |
| + DrawButton()                  |
+---------------------------------+
```

主类GomokuGame

代码块

```
+-----------------------------------------------------+
|                  class GomokuGame                   |
```

```
+-----------------------------------------------------+
| - gameMode: GameMode                                |
| - gameState: GameState                              |
| - currentPlayer: Piece                              |
| - board[BOARD_SIZE][BOARD_SIZE]: Piece              |
| - moveHistory: stack<pair<int, int>>                |
| - bgMusic: Music                                    |
| - placeSound: Sound                                 |
| - winSound: Sound                                   |
| - aiWinSound: Sound                                 |
| - musicPlaying: bool                                |
+-----------------------------------------------------+
| + Run()                                             |
| - InitBoard()                                       |
| - ClearBoard()                                      |
| - DrawBoard()                                       |
| - CheckWin(x: int, y: int): bool                    |
| - UndoMove()                                        |
| - EvaluateBoard(): int                              |
| - Minimax(depth: int, alpha: int, beta: int,        |
|           maximizingPlayer: bool): int              |
| - AIPlay(hardMode: bool)                            |
| - DrawGameUI()                                      |
| - DrawMenu()                                        |
| - Update()                                          |
| - Draw()                                            |
+-----------------------------------------------------+
```

关系图

代码块

```
+-----------------+         +-----------------------+
|     GameMode    |<--------/     GomokuGame        /
+-----------------+         +-----------------------+
        ^                     | - gameMode            |
        |                     +-----------------------+
+-----------------+               #
|    GameState    |<--------------+
+-----------------+               #
        ^                         #
        |                         v
+-----------------+         +-----------------------+
|     Piece       |<--------/     GomokuGame        /
```

```
+-----------------+          | - currentPlayer        |
                             | - board[15][15]        |
                             +------------------------+

                             +------------------------+
                             |        Button          |
                             +------------------------+
                             | - bounds               |
                             | - text                 |
                             | - isHovered            |
                             +------------------------+
                                        ^
                                        |
                             +------------------------+
                             |     GomokuGame         |
                             | - UI buttons           |
                             +------------------------+

                             +------------------------+
                             |        Music           |
                             +------------------------+
                             | - bgMusic              |
                             +------------------------+
                                        ^
                                        |
                             +------------------------+
                             |     GomokuGame         |
                             | - bgMusic              |
                             +------------------------+

                             +------------------------+
                             |        Sound           |
                             +------------------------+
                             | - placeSound           |
                             | - winSound             |
                             | - aiWinSound           |
                             +------------------------+
                                        ^
                                        |
                             +------------------------+
                             |     GomokuGame         |
                             | - sounds               |
                             +------------------------+

                             +----------------------------+
                             |  std::stack<std::pair<int, int>> |
                             +----------------------------+
```

```
                                    ^
                                    |
                          +-------------------------+
                          |     GomokuGame          |
                          | - moveHistory           |
                          +-------------------------+
```

六、代码思路概述

本项目采用面向对象的设计思想，使用 C++ 和 Raylib 图形库实现了一个支持玩家对战（PvP）和人机对战（PvE）的游戏系统。

程序核心是一个 `GomokuGame` 类，管理游戏状态、棋盘绘制、用户交互、AI 决策、音效播放等功能。游戏具有图形界面、按钮操作、悔棋功能和胜利判断机制，AI 部分分为简单模式（基于评估函数）和困难模式（使用 Minimax + Alpha-Beta 剪枝算法），具备一定的智能决策能力。

第一部分 头文件

代码块

```cpp
//****************第一部分 头文件****************
#include <iostream>
#include <raylib.h> //图形库
#include <vector>
#include <algorithm>
#include <climits> //// 提供 INT_MAX 和 INT_MIN 等常量
#include <stack>   //栈的数据结构,后续用于实现悔棋的功能.
```

第二部分 游戏常量定义,基础枚举类型,结构体定义

代码块

```cpp
//****************第二部分 游戏常量定义,基础枚举类型,结构体定义****************
// 游戏常量
const int BOARD_SIZE = 15; // 15x15棋盘
const int CELL_SIZE = 40;  // 每个格子的大小
const int PADDING = 50;    // 边缘距离
```

```
// 窗口宽高
const int WINDOW_WIDTH = 1000;
const int WINDOW_HEIGHT = 660;

// 颜色定义
const Color BOARD_COLOR = {210, 180, 140, 255}; // 棋盘背景色,木色
const Color LINE_COLOR = BLACK;                 // 线的颜色
const Color TEXT_COLOR = BLACK;                 // 文字的颜色
const Color UI_BG_COLOR = {136, 136, 136, 255}; // 右侧UI的颜色

// 棋子类型
enum Piece
{
    PIECE_EMPTY = 0, // 空
    PIECE_BLACK = 1, // 黑子
    PIECE_WHITE = 2  // 白子
};

// 游戏状态
enum GameState
{
    STATE_MENU,      // 菜单
    STATE_PLAYING,   // 游戏进行中
    STATE_BLACK_WIN, // 黑子获胜
    STATE_WHITE_WIN  // 白子获胜
};

// 三种游戏模式
enum GameMode
{
    MODE_PVP,      // 玩家对战
    MODE_PVE_EASY, // easy AI
    MODE_PVE_HARD  // hard AI
};

// 按钮结构体
struct Button
{
    Rectangle bounds; // 按钮的边界
    const char *text; // 文本
    bool isHovered;   // 是否悬停
};
```

第三部分 GomokuGame主类

析构函数

释放资源

代码块

```
    // 析构函数,释放资源
    ~GomokuGame()
    {
        UnloadMusicStream(bgMusic);
        UnloadSound(placeSound);
        UnloadSound(winSound);
        UnloadSound(aiWinSound);
        CloseAudioDevice();
    }
```

游戏主循环函数

代码块

```
    // 游戏主循环函数
    void Run()
    {
        InitWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Gomoku Game"); // 创建窗口
        SetTargetFPS(60);                                       // 设置帧率60

        // 游戏住循环
        while (!WindowShouldClose())
        {
            // 更新游戏状态
            Update();
            Draw();
        }

        // 关闭窗口
        CloseWindow();
    }
```

游戏大体部分逻辑(棋盘,按钮)

创建,绘制按钮

代码块

```
    // 创建一个按钮变量
    Button CreateButton(float x, float y, float width, float height, const char *text)
    {
        return (Button){{x, y, width, height}, text, false};
    }

    // 绘制按钮
    void DrawButton(Button *button)
    {
        Color btnColor = button->isHovered ? SKYBLUE : LIGHTGRAY;
        DrawRectangleRec(button->bounds, btnColor);
        DrawRectangleLinesEx(button->bounds, 2, DARKGRAY);
        int textWidth = MeasureText(button->text, 20);
        DrawText(button->text,
                 button->bounds.x + (button->bounds.width - textWidth) / 2,
                 button->bounds.y + 10,
                 20, DARKGRAY);
    }
```

初始化,清空棋盘

代码块

```
    // 初始化棋盘,此时没有任何棋子
    void InitBoard()
    {
        while (!moveHistory.empty())
            moveHistory.pop();
        for (int y = 0; y < BOARD_SIZE; y++)
        {
            for (int x = 0; x < BOARD_SIZE; x++)
            {
                board[y][x] = PIECE_EMPTY;
            }
        }
        currentPlayer = PIECE_BLACK;
    }

    // 清空棋盘,调用InitBoard()函数即可
    void ClearBoard()
    {
        InitBoard();
        gameState = STATE_PLAYING;
```

```
    }
```

绘制棋盘

代码块

```
    // 绘制棋盘及其元素
    void DrawBoard()
    {
        // 棋盘背景
        DrawRectangle(
            PADDING - 10,
            PADDING - 10,
            (BOARD_SIZE - 1) * CELL_SIZE + 20,
            (BOARD_SIZE - 1) * CELL_SIZE + 20,
            BOARD_COLOR);

        // 网格线
        for (int i = 0; i < BOARD_SIZE; i++)
        {
            DrawLineEx(
                Vector2{(float)PADDING, (float)(PADDING + i * CELL_SIZE)},
                Vector2{(float)(PADDING + (BOARD_SIZE - 1) * CELL_SIZE),
(float)(PADDING + i * CELL_SIZE)},
                1.5f, LINE_COLOR);
            DrawLineEx(
                Vector2{(float)(PADDING + i * CELL_SIZE), (float)PADDING},
                Vector2{(float)(PADDING + i * CELL_SIZE), (float)(PADDING +
(BOARD_SIZE - 1) * CELL_SIZE)},
                1.5f, LINE_COLOR);
        }

        // 星位点(五子棋棋盘的标准布局)
        const int starPoints[5][2] = {{3, 3}, {11, 3}, {3, 11}, {11, 11}, {7,
7}};
        for (const auto &point : starPoints)
        {
            DrawCircle(
                PADDING + point[0] * CELL_SIZE,
                PADDING + point[1] * CELL_SIZE,
                5, LINE_COLOR);
        }

        // 放置棋子
```

```
        for (int y = 0; y < BOARD_SIZE; y++)
        {
            for (int x = 0; x < BOARD_SIZE; x++)
            {
                if (board[y][x] == PIECE_BLACK)
                {
                    DrawCircle(
                        PADDING + x * CELL_SIZE,
                        PADDING + y * CELL_SIZE,
                        CELL_SIZE / 2 - 2, BLACK);
                }
                else if (board[y][x] == PIECE_WHITE)
                {
                    DrawCircle(
                        PADDING + x * CELL_SIZE,
                        PADDING + y * CELL_SIZE,
                        CELL_SIZE / 2 - 2, WHITE);
                    DrawCircleLines(
                        PADDING + x * CELL_SIZE,
                        PADDING + y * CELL_SIZE,
                        CELL_SIZE / 2 - 2, BLACK);
                }
            }
        }
    }

```

判断是否有五子连珠

代码块

```
    // 检查是否有五子连珠,检查黑子和白子在四个方向(横,竖,左斜,右斜)
    bool CheckWin(int x, int y)
    {
        if (board[y][x] == PIECE_EMPTY)
            return false;

        const int dirs[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};
        for (const auto &dir : dirs)
        {
            int count = 1;
            int dx = dir[0], dy = dir[1];

            for (int i = 1; i < 5; i++)
            {
                int nx = x + dx * i, ny = y + dy * i;
```

```
                if (nx >= BOARD_SIZE || ny >= BOARD_SIZE || board[ny][nx] != board[y][x])
                    break;
                count++;
            }

            for (int i = 1; i < 5; i++)
            {
                int nx = x - dx * i, ny = y - dy * i;
                if (nx < 0 || ny < 0 || board[ny][nx] != board[y][x])
                    break;
                count++;
            }

            // 如果同一个方向等于五个字,就直接返回
            if (count >= 5)
                return true;
        }

        return false;
    }
```

悔棋功能

代码块

```
    // 悔棋函数,使用栈的数据结构实现,撤销最近的一步棋
    void UndoMove()
    {
        if (moveHistory.empty())
            return;

        auto lastMove = moveHistory.top();
        moveHistory.pop();
        board[lastMove.second][lastMove.first] = PIECE_EMPTY;
        currentPlayer = (currentPlayer == PIECE_BLACK) ? PIECE_WHITE : PIECE_BLACK;

        // 如果撤回的是AI的轮次,则回退两步棋(AI一次,人类一次),保证悔棋完成后是人类的轮次
        if ((gameMode == MODE_PVE_EASY || gameMode == MODE_PVE_HARD) &&
            currentPlayer == PIECE_WHITE && !moveHistory.empty())
        {
            lastMove = moveHistory.top();
            moveHistory.pop();
            board[lastMove.second][lastMove.first] = PIECE_EMPTY;
            currentPlayer = PIECE_BLACK;
```

```
        }
    }
```

AI下棋算法(本项目的核心功能⭐)

评估函数

评估函数,评估棋盘的得分,从而选择下一步棋的最佳位置

代码块

```
    int EvaluateBoard()
    {
        int score = 0;
        for (int y = 0; y < BOARD_SIZE; y++)
        {
            for (int x = 0; x < BOARD_SIZE; x++)
            {
                if (board[y][x] != PIECE_EMPTY)
                {
                    const int dirs[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};

                    for (const auto &dir : dirs)
                    {
                        int dx = dir[0], dy = dir[1];
                        int count = 1;
                        bool blocked = false;

                        for (int i = 1; i < 5; i++)
                        {
                            int nx = x + dx * i, ny = y + dy * i;
                            if (nx < 0 || nx >= BOARD_SIZE || ny < 0 || ny >=
BOARD_SIZE)
                            {
                                blocked = true;
                                break;
                            }
                            if (board[ny][nx] == board[y][x])
                                count++;
                            else if (board[ny][nx] == PIECE_EMPTY)
                                break;
                            else
                            {
                                blocked = true;
                                break;
```

```
                            }
                        }

                        if (count >= 5)
                            return (board[y][x] == PIECE_BLACK) ? INT_MAX :
    INT_MIN;

                        int value = 0;
                        switch (count)
                        {
                        case 4:
                            value = (blocked) ? 500 : 2000;
                            break;
                        case 3:
                            value = (blocked) ? 100 : 500;
                            break;
                        case 2:
                            value = (blocked) ? 10 : 50;
                            break;
                        case 1:
                            value = 1;
                            break;
                        }

                        if (board[y][x] == PIECE_BLACK)
                            score += value;
                        else
                            score -= value;
                    }
                }
            }
        }
        return score;
    }

```

获取棋盘上可下的位置

代码块

```
    // 获取当前棋盘上所有可下的位置
    std::vector<std::pair<int, int>> GetPossibleMoves()
    {
        std::vector<std::pair<int, int>> moves;
```

```
5           for (int y = 0; y < BOARD_SIZE; y++)
6           {
7               for (int x = 0; x < BOARD_SIZE; x++)
8               {
9                   if (board[y][x] != PIECE_EMPTY)
10                  {
11                      for (int dy = -1; dy <= 1; dy++)
12                      {
13                          for (int dx = -1; dx <= 1; dx++)
14                          {
15                              if (dx == 0 && dy == 0)
16                                  continue;
17
18                              int nx = x + dx, ny = y + dy;
19                              if (nx >= 0 && nx < BOARD_SIZE && ny >= 0 && ny <
    BOARD_SIZE &&
20                                  board[ny][nx] == PIECE_EMPTY)
21                              {
22                                  bool exists = false;
23                                  for (const auto &move : moves)
24                                  {
25                                      if (move.first == nx && move.second == ny)
26                                      {
27                                          exists = true;
28                                          break;
29                                      }
30                                  }
31                                  if (!exists)
32                                      moves.emplace_back(nx, ny);
33                              }
34                          }
35                      }
36                  }
37              }
38          }
39          if (moves.empty())
40              moves.emplace_back(BOARD_SIZE / 2, BOARD_SIZE / 2);
41          return moves;
42      }
```

`Minmax` + `Alpha-Beta` 剪枝算法

代码块

```cpp
// 递归实现最小最大算法,加上Alpha-Beta 剪枝，用于AI决策
int Minimax(int depth, int alpha, int beta, bool maximizingPlayer)
{
    if (depth == 0)
        return EvaluateBoard();

    auto moves = GetPossibleMoves();
    if (moves.empty())
        return EvaluateBoard();

    if (maximizingPlayer)
    {
        int maxEval = INT_MIN;
        for (const auto &move : moves)
        {
            int x = move.first, y = move.second;
            board[y][x] = PIECE_BLACK;
            if (CheckWin(x, y))
            {
                board[y][x] = PIECE_EMPTY;
                return INT_MAX;
            }
            int eval = Minimax(depth - 1, alpha, beta, false);
            board[y][x] = PIECE_EMPTY;
            maxEval = std::max(maxEval, eval);
            alpha = std::max(alpha, eval);
            if (beta <= alpha)
                break;
        }
        return maxEval;
    }
    else
    {
        int minEval = INT_MAX;
        for (const auto &move : moves)
        {
            int x = move.first, y = move.second;
            board[y][x] = PIECE_WHITE;
            if (CheckWin(x, y))
            {
                board[y][x] = PIECE_EMPTY;
                return INT_MIN;
            }
            int eval = Minimax(depth - 1, alpha, beta, true);
            board[y][x] = PIECE_EMPTY;
            minEval = std::min(minEval, eval);
            beta = std::min(beta, eval);
```

```
                if (beta <= alpha)
                    break;
            }
            return minEval;
        }
    }
```

AI落子逻辑

代码块

```
    void AIPlay(bool hardMode)
    {
        auto moves = GetPossibleMoves();
        if (moves.empty())
            return;

        int bestScore = (currentPlayer == PIECE_BLACK) ? INT_MIN : INT_MAX;
        std::pair<int, int> bestMove = moves[0];

        for (const auto &move : moves)
        {
            int x = move.first, y = move.second;
            board[y][x] = currentPlayer;
            if (CheckWin(x, y))
            {
                board[y][x] = PIECE_EMPTY;
                bestMove = move;
                break;
            }
            int score;
            if (hardMode)
            {
                score = Minimax(3, INT_MIN, INT_MAX, currentPlayer ==
PIECE_WHITE);
            }
            else
            {
                score = EvaluateBoard();
                if (currentPlayer == PIECE_WHITE)
                    score = -score;
            }
            board[y][x] = PIECE_EMPTY;

```

```
            if (currentPlayer == PIECE_BLACK)
            {
                if (score > bestScore)
                {
                    bestScore = score;
                    bestMove = move;
                }
            }
            else
            {
                if (score < bestScore)
                {
                    bestScore = score;
                    bestMove = move;
                }
            }
        }

        int x = bestMove.first, y = bestMove.second;
        board[y][x] = currentPlayer;
        moveHistory.push(bestMove);
        PlaySound(placeSound);

        if (CheckWin(x, y))
        {
            gameState = (currentPlayer == PIECE_BLACK) ? STATE_BLACK_WIN :
STATE_WHITE_WIN;
            if (gameMode == MODE_PVP || currentPlayer == PIECE_BLACK)
            {
                PlaySound(winSound);
            }
            else
            {
                PlaySound(aiWinSound);
            }
        }
        else
        {
            currentPlayer = (currentPlayer == PIECE_BLACK) ? PIECE_WHITE :
PIECE_BLACK;
        }
    }
```

绘制游戏UI

代码块

```
    // 绘制游戏右侧UI,边栏部分
    void DrawGameUI()
    {
        DrawRectangle(700, 0, 300, WINDOW_HEIGHT, UI_BG_COLOR);
        const char *playerText = (currentPlayer == PIECE_BLACK) ? "Now: Black"
: "Now: White";
        DrawText(playerText, 720, 20, 24, TEXT_COLOR);

        const char *modeText = "";
        switch (gameMode)
        {
        case MODE_PVP:
            modeText = "Pair Game";
            break;
        case MODE_PVE_EASY:
            modeText = "AI game(Easy)";
            break;
        case MODE_PVE_HARD:
            modeText = "AI Game(Hard)";
            break;
        }
        DrawText(modeText, 720, 60, 20, TEXT_COLOR);

        // 悔棋按钮
        Button btnUndo = CreateButton(720, 100, 150, 40, "Undo");
        btnUndo.isHovered = CheckCollisionPointRec(GetMousePosition(),
btnUndo.bounds);
        DrawButton(&btnUndo);

        // 清空棋盘按钮
        Button btnClear = CreateButton(720, 150, 150, 40, "Clear Board");
        btnClear.isHovered = CheckCollisionPointRec(GetMousePosition(),
btnClear.bounds);
        DrawButton(&btnClear);

        // 返回菜单按钮
        Button btnMenu = CreateButton(720, 200, 150, 40, "Back to Menu");
        btnMenu.isHovered = CheckCollisionPointRec(GetMousePosition(),
btnMenu.bounds);
        DrawButton(&btnMenu);

        // 音乐控制按钮
        Button btnMusic = CreateButton(720, 250, 150, 40, musicPlaying ? "Mute"
 : "Play Music");
```

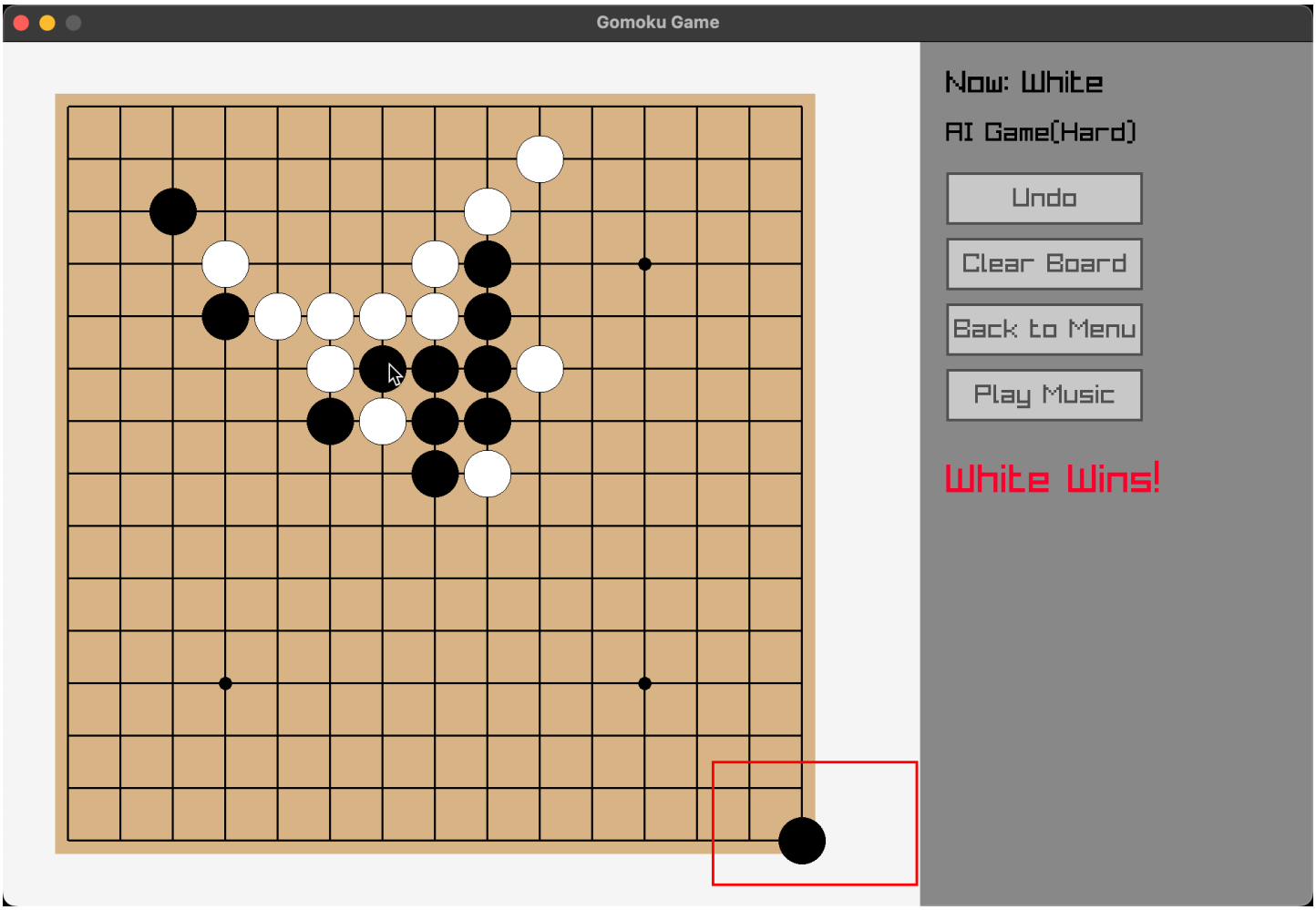
```
        btnMusic.isHovered = CheckCollisionPointRec(GetMousePosition(), btnMusic.bounds);
        DrawButton(&btnMusic);

        if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON))
        {
            if (btnUndo.isHovered)
            {
                UndoMove();
            }
            else if (btnClear.isHovered)
            {
                ClearBoard();
            }
            else if (btnMenu.isHovered)
            {
                gameState = STATE_MENU;
            }
            else if (btnMusic.isHovered)
            {
                musicPlaying = !musicPlaying;
                if (musicPlaying)
                {
                    PlayMusicStream(bgMusic);
                }
                else
                {
                    StopMusicStream(bgMusic);
                }
            }
        }

        if (gameState == STATE_BLACK_WIN)
        {
            DrawText("Black Wins!", 720, 320, 30, RED);
        }
        else if (gameState == STATE_WHITE_WIN)
        {
            DrawText("White Wins!", 720, 320, 30, RED);
        }
    }

    // 绘制游戏主菜单界面
```

七、项目测试

1. 基础功能测试

测试项	测试方法	预期结果	测试结果
落子功能	点击棋盘交叉点	正确显示黑白棋子	正常
胜负判断	制造五连珠局面	正确识别胜负	正常
悔棋功能	点击Undo按钮	撤销上一步操作	正常
AI对战	与AI对弈	AI能做出合理应对	正常

2. 边界测试,棋盘边缘落子



3. 连续悔棋测试 ✅
4. 音乐开关测试 ✅

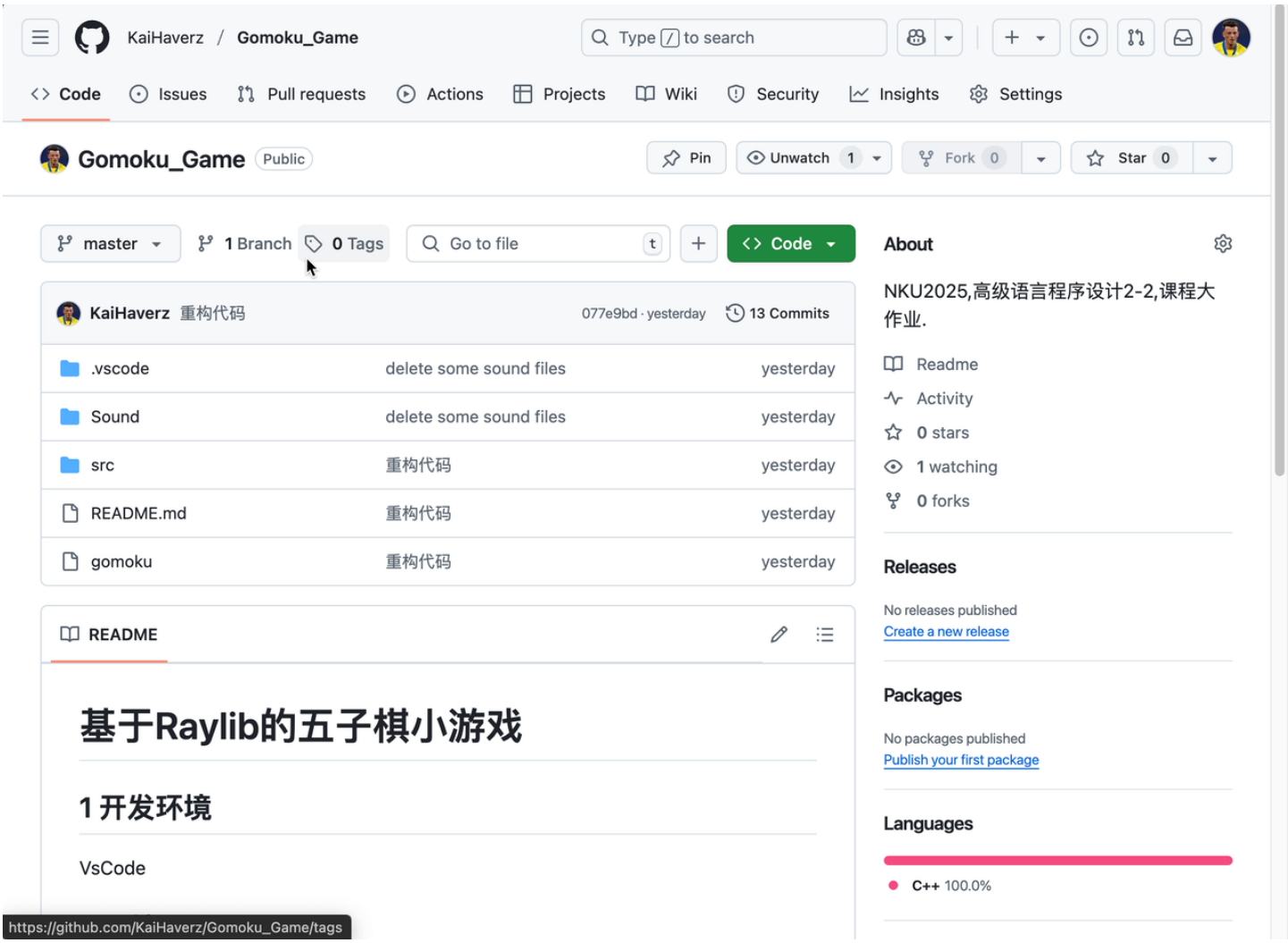
所有基本功能测试通过，AI对战也有不错表现

八、项目总结

1. 本项目基于Raylib开发了一个功能完善的五子棋游戏，使用了本学期所学的面向对象编程思想, 实现了双人对战(PVP)和人机对战(PVE)两种模式.
2. 其中AI采用评估函数(简单模式)和Minimax算法结合Alpha-Beta剪枝(困难模式)进行决策。
3. 游戏具备完整的UI界面、音效系统、悔棋功能和胜负判定机制，通过面向对象设计将游戏逻辑、界面和音效封装在GomokuGame类中。
4. 项目亮点在于AI算法的实现和良好的用户体验.
5. 同时我也积累了一次大项目的开发经验,评估函数中的权重设置反复调整,处理各种边界情况
6. 同时使用Git,Github管理一个较大的长期开发项目,学到了一些项目版本控制的知识.
7. 项目也有一些待完善的地方,比如说AI算法,界面的美观程度等等

九、源代码

- 详见Github仓库: https://github.com/KaiHaverz/Gomoku_Game



- 提交次数: 15
  - 10995++
  - 9495--