



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

二、开发环境

三、项目概述

项目功能

效果预览

四、课题要求

五、主要流程

六、类图

枚举类

结构体Button

主类GomokuGame

关系图

六、代码思路概述

第一部分 头文件

第二部分 游戏常量定义,基础枚举类型,结构体定义

第三部分 GomokuGame主类

析构函数

游戏主循环函数

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

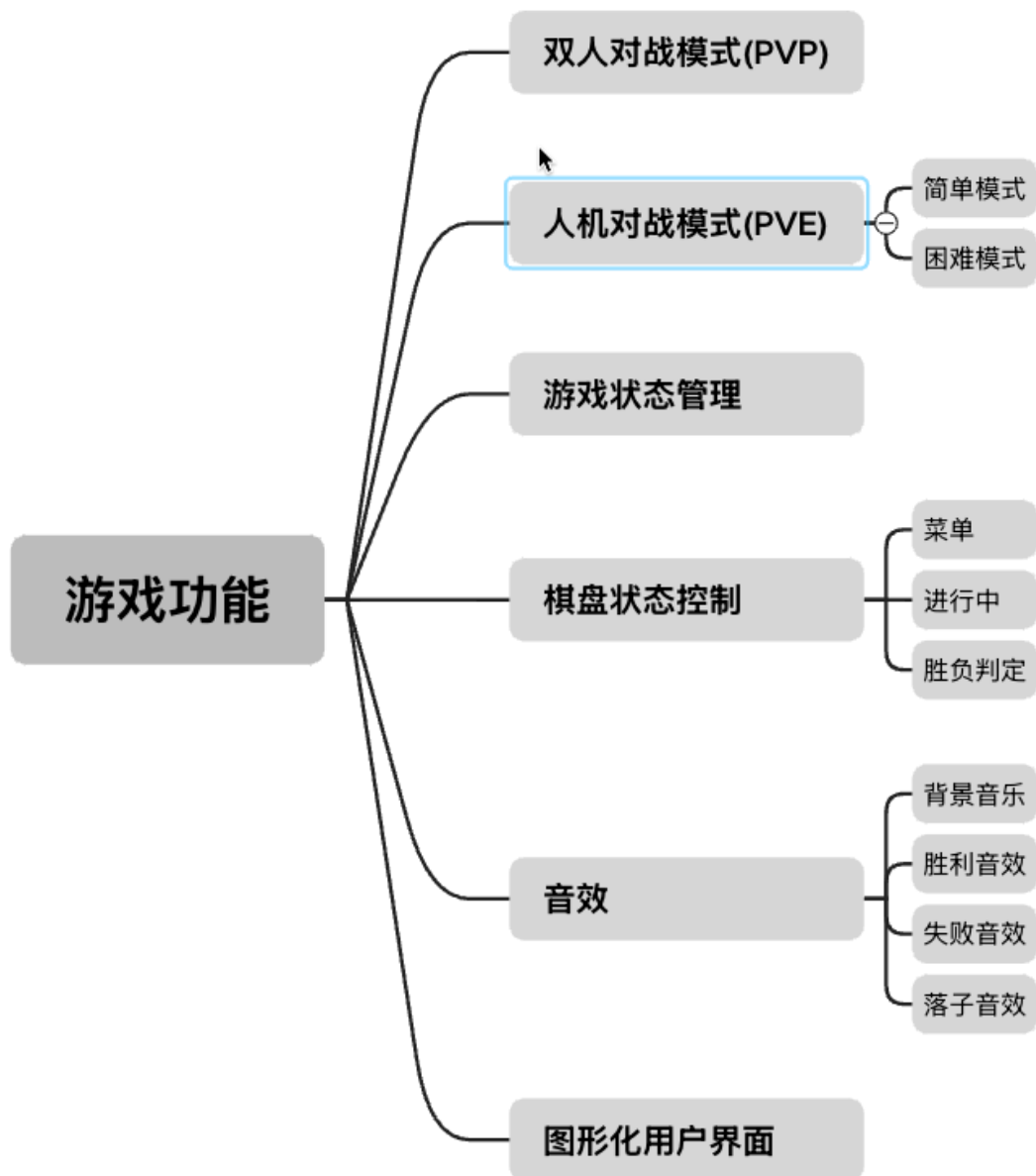
- IDE: VsCode
- 图形化工具: Raylib

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

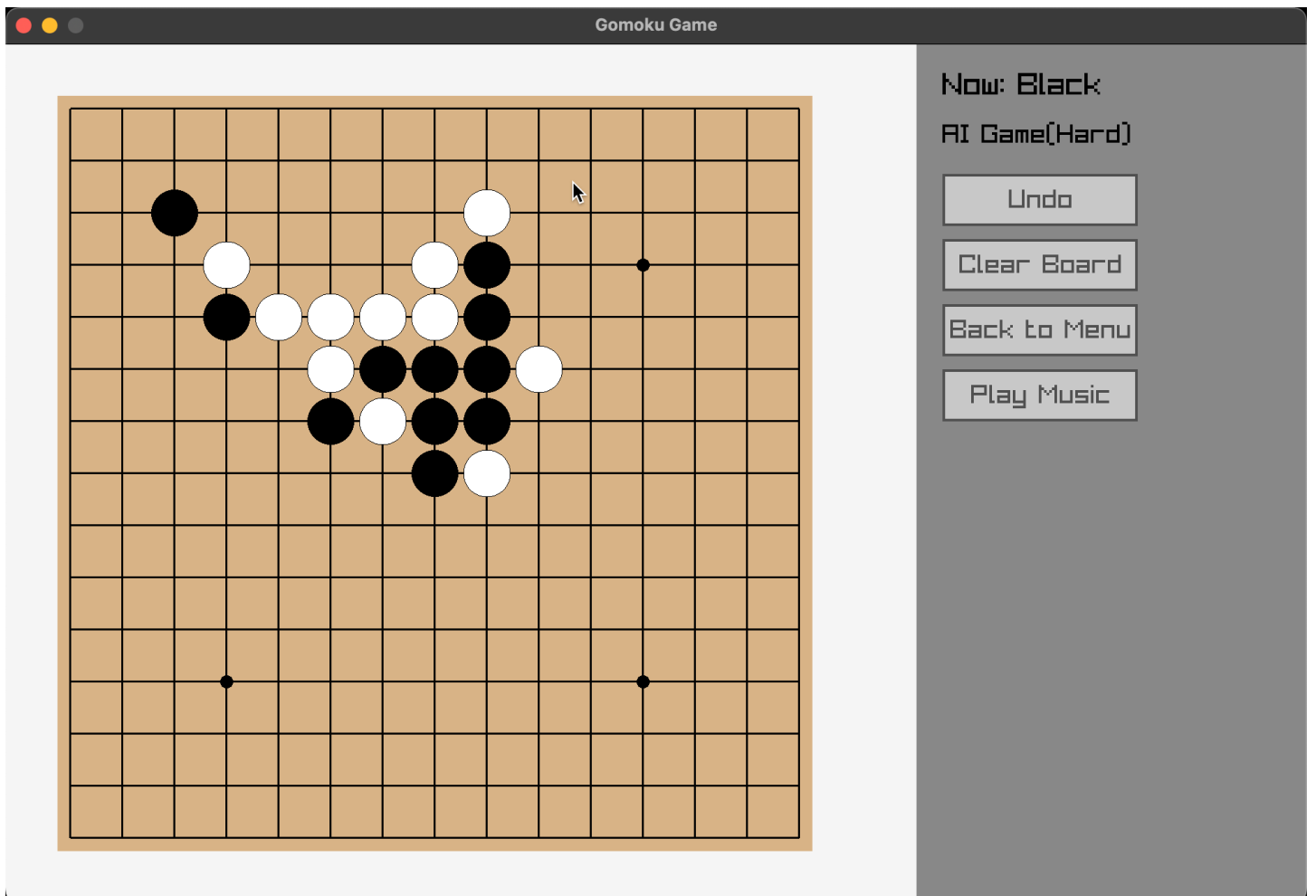


效果预览

主页菜单:



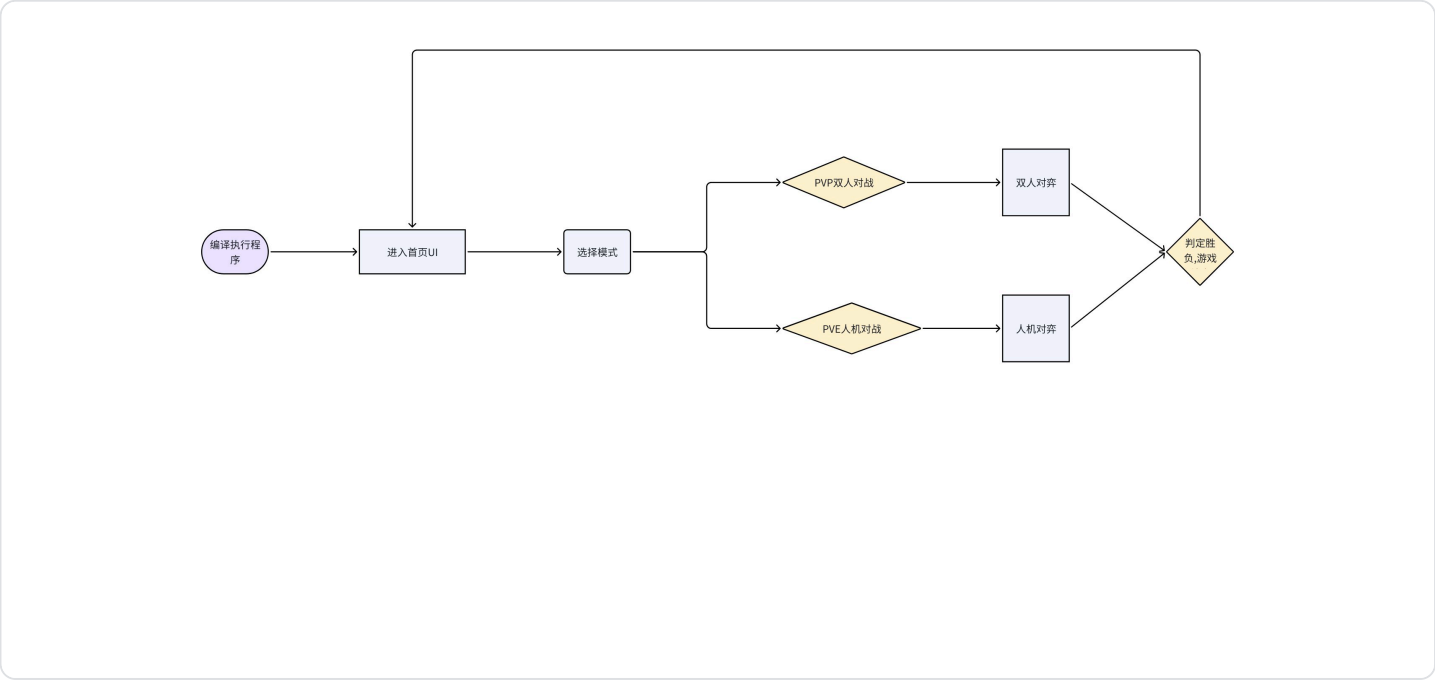
游戏界面:



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

```
1  +-----+
2  |      enum Piece      |
3  +-----+
4  |  PIECE_EMPTY = 0      |
5  |  PIECE_BLACK  = 1      |
```

```

6  |  PIECE_WHITE = 2  |
7  +-----+
8
9  +-----+
10 |  enum GameState  |
11 +-----+
12 |  STATE_MENU      |
13 |  STATE_PLAYING   |
14 |  STATE_BLACK_WIN |
15 |  STATE_WHITE_WIN |
16 +-----+
17
18 +-----+
19 |  enum GameMode    |
20 +-----+
21 |  MODE_PVP         |
22 |  MODE_PVE_EASY    |
23 |  MODE_PVE_HARD    |
24 +-----+

```

结构体Button

代码块

```

1  +-----+
2  |          struct Button          |
3  +-----+
4  | - bounds: Rectangle              |
5  | - text: const char*              |
6  | - isHovered: bool               |
7  +-----+
8  | + CreateButton()                 |
9  | + DrawButton()                  |
10 +-----+

```

主类GomokuGame

代码块

```

1  +-----+
2  |          class GomokuGame          |

```

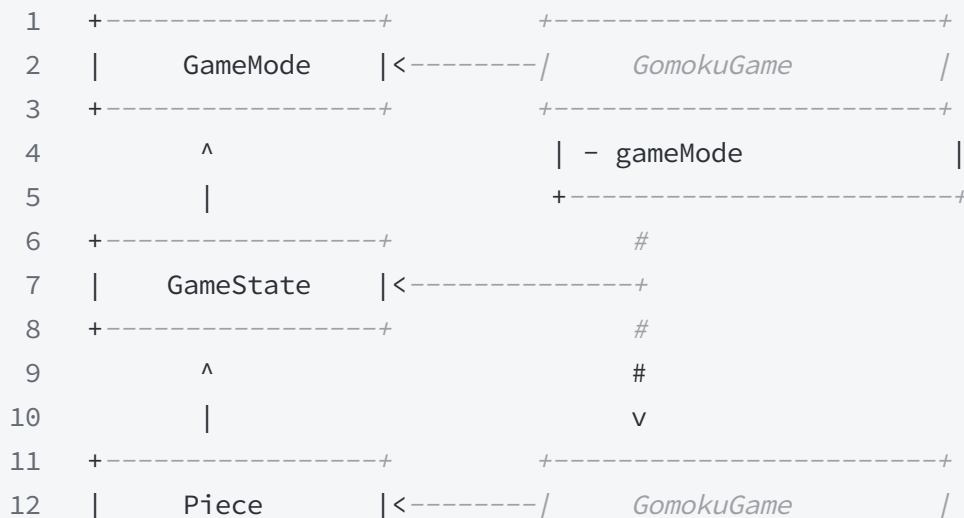
```

3  +-----+
4  | - gameMode: GameMode |
5  | - gameState: GameState |
6  | - currentPlayer: Piece |
7  | - board[BOARD_SIZE][BOARD_SIZE]: Piece |
8  | - moveHistory: stack<pair<int, int>> |
9  | - bgMusic: Music |
10 | - placeSound: Sound |
11 | - winSound: Sound |
12 | - aiWinSound: Sound |
13 | - musicPlaying: bool |
14 +-----+
15 | + Run() |
16 | - InitBoard() |
17 | - ClearBoard() |
18 | - DrawBoard() |
19 | - CheckWin(x: int, y: int): bool |
20 | - UndoMove() |
21 | - EvaluateBoard(): int |
22 | - Minimax(depth: int, alpha: int, beta: int, |
23 |           maximizingPlayer: bool): int |
24 | - AIPlay(hardMode: bool) |
25 | - DrawGameUI() |
26 | - DrawMenu() |
27 | - Update() |
28 | - Draw() |
29 +-----+

```

关系图

代码块




```

13  +-----+ / - currentPlayer /
14  | - board[15][15] |
15  +-----+
16
17  +-----+
18  |           Button           |
19  +-----+
20  | - bounds                   |
21  | - text                     |
22  | - isHovered               |
23  +-----+
24  |           ^               |
25  |           |               |
26  +-----+
27  |           GomokuGame       |
28  | - UI buttons               |
29  +-----+
30
31  +-----+
32  |           Music            |
33  +-----+
34  | - bgMusic                  |
35  +-----+
36  |           ^               |
37  |           |               |
38  +-----+
39  |           GomokuGame       |
40  | - bgMusic                  |
41  +-----+
42
43  +-----+
44  |           Sound            |
45  +-----+
46  | - placeSound               |
47  | - winSound                 |
48  | - aiWinSound               |
49  +-----+
50  |           ^               |
51  |           |               |
52  +-----+
53  |           GomokuGame       |
54  | - sounds                   |
55  +-----+
56
57  +-----+
58  | std::stack<std::pair<int, int>> |
59  +-----+

```

```

60                                     ^
61                                     |
62      +-----+
63      |      GomokuGame      |
64      | - moveHistory        |
65      +-----+

```

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

```

1  //*****第一部分 头文件*****
2  #include <iostream>
3  #include <raylib.h> //图形库
4  #include <vector>
5  #include <algorithm>
6  #include <climits> //提供 INT_MAX 和 INT_MIN 等常量
7  #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

```

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

```

```

6 // 窗口宽高
7 const int WINDOW_WIDTH = 1000;
8 const int WINDOW_HEIGHT = 660;
9
10 // 颜色定义
11 const Color BOARD_COLOR = {210, 180, 140, 255}; // 棋盘背景色,木色
12 const Color LINE_COLOR = BLACK; // 线的颜色
13 const Color TEXT_COLOR = BLACK; // 文字的颜色
14 const Color UI_BG_COLOR = {136, 136, 136, 255}; // 右侧UI的颜色
15
16 // 棋子类型
17 enum Piece
18 {
19     PIECE_EMPTY = 0, // 空
20     PIECE_BLACK = 1, // 黑子
21     PIECE_WHITE = 2 // 白子
22 };
23
24 // 游戏状态
25 enum GameState
26 {
27     STATE_MENU, // 菜单
28     STATE_PLAYING, // 游戏进行中
29     STATE_BLACK_WIN, // 黑子获胜
30     STATE_WHITE_WIN // 白子获胜
31 };
32
33 // 三种游戏模式
34 enum GameMode
35 {
36     MODE_PVP, // 玩家对战
37     MODE_PVE_EASY, // easy AI
38     MODE_PVE_HARD // hard AI
39 };
40
41 // 按钮结构体
42 struct Button
43 {
44     Rectangle bounds; // 按钮的边界
45     const char *text; // 文本
46     bool isHovered; // 是否悬停
47 };

```

第三部分 GomokuGame主类

析构函数

释放资源

代码块

```
1      // 析构函数,释放资源
2      ~GomokuGame()
3      {
4          UnloadMusicStream(bgMusic);
5          UnloadSound(placeSound);
6          UnloadSound(winSound);
7          UnloadSound(aiWinSound);
8          CloseAudioDevice();
9      }
```

游戏主循环函数

代码块

```
1      // 游戏主循环函数
2      void Run()
3      {
4          InitWindow(WINDOW_WIDTH, WINDOW_HEIGHT, "Gomoku Game"); // 创建窗口
5          SetTargetFPS(60); // 设置帧率60
6
7          // 游戏住循环
8          while (!WindowShouldClose())
9          {
10             // 更新游戏状态
11             Update();
12             Draw();
13         }
14
15         // 关闭窗口
16         CloseWindow();
17     }
```

游戏大体部分逻辑(棋盘,按钮)

创建,绘制按钮

代码块

```

1      // 创建一个按钮变量
2      Button CreateButton(float x, float y, float width, float height, const char
      *text)
3      {
4          return (Button){x, y, width, height, text, false};
5      }
6
7      // 绘制按钮
8      void DrawButton(Button *button)
9      {
10         Color btnColor = button->isHovered ? SKYBLUE : LIGHTGRAY;
11         DrawRectangleRec(button->bounds, btnColor);
12         DrawRectangleLinesEx(button->bounds, 2, DARKGRAY);
13         int textWidth = MeasureText(button->text, 20);
14         DrawText(button->text,
15                 button->bounds.x + (button->bounds.width - textWidth) / 2,
16                 button->bounds.y + 10,
17                 20, DARKGRAY);
18     }

```

初始化,清空棋盘

代码块

```

1      // 初始化棋盘,此时没有任何棋子
2      void InitBoard()
3      {
4          while (!moveHistory.empty())
5              moveHistory.pop();
6          for (int y = 0; y < BOARD_SIZE; y++)
7          {
8              for (int x = 0; x < BOARD_SIZE; x++)
9              {
10                 board[y][x] = PIECE_EMPTY;
11             }
12         }
13         currentPlayer = PIECE_BLACK;
14     }
15
16     // 清空棋盘,调用InitBoard()函数即可
17     void ClearBoard()
18     {
19         InitBoard();
20         gameState = STATE_PLAYING;

```

```
21     }
```

绘制棋盘

代码块

```
1      // 绘制棋盘及其元素
2      void DrawBoard()
3      {
4          // 棋盘背景
5          DrawRectangle(
6              PADDING - 10,
7              PADDING - 10,
8              (BOARD_SIZE - 1) * CELL_SIZE + 20,
9              (BOARD_SIZE - 1) * CELL_SIZE + 20,
10             BOARD_COLOR);
11
12         // 网格线
13         for (int i = 0; i < BOARD_SIZE; i++)
14         {
15             DrawLineEx(
16                 Vector2{(float)PADDING, (float)(PADDING + i * CELL_SIZE)},
17                 Vector2{(float)(PADDING + (BOARD_SIZE - 1) * CELL_SIZE),
18                     (float)(PADDING + i * CELL_SIZE)},
19                 1.5f, LINE_COLOR);
20             DrawLineEx(
21                 Vector2{(float)(PADDING + i * CELL_SIZE), (float)PADDING},
22                 Vector2{(float)(PADDING + i * CELL_SIZE), (float)(PADDING +
23                     (BOARD_SIZE - 1) * CELL_SIZE)},
24                 1.5f, LINE_COLOR);
25         }
26
27         // 星位点(五子棋棋盘的标准布局)
28         const int starPoints[5][2] = {{3, 3}, {11, 3}, {3, 11}, {11, 11}, {7,
29             7}};
30
31         for (const auto &point : starPoints)
32         {
33             DrawCircle(
34                 PADDING + point[0] * CELL_SIZE,
35                 PADDING + point[1] * CELL_SIZE,
36                 5, LINE_COLOR);
37         }
38
39         // 放置棋子
```

```

36         for (int y = 0; y < BOARD_SIZE; y++)
37         {
38             for (int x = 0; x < BOARD_SIZE; x++)
39             {
40                 if (board[y][x] == PIECE_BLACK)
41                 {
42                     DrawCircle(
43                         PADDING + x * CELL_SIZE,
44                         PADDING + y * CELL_SIZE,
45                         CELL_SIZE / 2 - 2, BLACK);
46                 }
47                 else if (board[y][x] == PIECE_WHITE)
48                 {
49                     DrawCircle(
50                         PADDING + x * CELL_SIZE,
51                         PADDING + y * CELL_SIZE,
52                         CELL_SIZE / 2 - 2, WHITE);
53                     DrawCircleLines(
54                         PADDING + x * CELL_SIZE,
55                         PADDING + y * CELL_SIZE,
56                         CELL_SIZE / 2 - 2, BLACK);
57                 }
58             }
59         }
60     }
61

```

判断是否有五子连珠

代码块

```

1      // 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)
2      bool CheckWin(int x, int y)
3      {
4          if (board[y][x] == PIECE_EMPTY)
5              return false;
6
7          const int dirs[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};
8          for (const auto &dir : dirs)
9          {
10             int count = 1;
11             int dx = dir[0], dy = dir[1];
12
13             for (int i = 1; i < 5; i++)
14             {
15                 int nx = x + dx * i, ny = y + dy * i;

```

```

16         if (nx >= BOARD_SIZE || ny >= BOARD_SIZE || board[ny][nx] !=
board[y][x])
17             break;
18         count++;
19     }
20
21     for (int i = 1; i < 5; i++)
22     {
23         int nx = x - dx * i, ny = y - dy * i;
24         if (nx < 0 || ny < 0 || board[ny][nx] != board[y][x])
25             break;
26         count++;
27     }
28
29     // 如果同一个方向等于五个字,就直接返回
30     if (count >= 5)
31         return true;
32     }
33
34     return false;
35 }

```

悔棋功能

代码块

```

1    // 悔棋函数,使用栈的数据结构实现,撤销最近的一步棋
2    void UndoMove()
3    {
4        if (moveHistory.empty())
5            return;
6
7        auto lastMove = moveHistory.top();
8        moveHistory.pop();
9        board[lastMove.second][lastMove.first] = PIECE_EMPTY;
10       currentPlayer = (currentPlayer == PIECE_BLACK) ? PIECE_WHITE :
PIECE_BLACK;
11
12       // 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
13       if ((gameMode == MODE_PVE_EASY || gameMode == MODE_PVE_HARD) &&
currentPlayer == PIECE_WHITE && !moveHistory.empty())
14       {
15           lastMove = moveHistory.top();
16           moveHistory.pop();
17           board[lastMove.second][lastMove.first] = PIECE_EMPTY;
18           currentPlayer = PIECE_BLACK;
19       }

```



```
20     }
21 }
```

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

```
1     int EvaluateBoard()
2     {
3         int score = 0;
4         for (int y = 0; y < BOARD_SIZE; y++)
5         {
6             for (int x = 0; x < BOARD_SIZE; x++)
7             {
8                 if (board[y][x] != PIECE_EMPTY)
9                 {
10                     const int dirs[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};
11
12                     for (const auto &dir : dirs)
13                     {
14                         int dx = dir[0], dy = dir[1];
15                         int count = 1;
16                         bool blocked = false;
17
18                         for (int i = 1; i < 5; i++)
19                         {
20                             int nx = x + dx * i, ny = y + dy * i;
21                             if (nx < 0 || nx >= BOARD_SIZE || ny < 0 || ny >=
22                                 BOARD_SIZE)
23                             {
24                                 blocked = true;
25                                 break;
26                             }
27                             if (board[ny][nx] == board[y][x])
28                                 count++;
29                             else if (board[ny][nx] == PIECE_EMPTY)
30                                 break;
31                             else
32                             {
33                                 blocked = true;
34                                 break;
35                             }
36                         }
37                     }
38                 }
39             }
40         }
41         return score;
42     }
```

```

34         }
35     }
36
37     if (count >= 5)
38         return (board[y][x] == PIECE_BLACK) ? INT_MAX :
INT_MIN;
39
40     int value = 0;
41     switch (count)
42     {
43     case 4:
44         value = (blocked) ? 500 : 2000;
45         break;
46     case 3:
47         value = (blocked) ? 100 : 500;
48         break;
49     case 2:
50         value = (blocked) ? 10 : 50;
51         break;
52     case 1:
53         value = 1;
54         break;
55     }
56
57     if (board[y][x] == PIECE_BLACK)
58         score += value;
59     else
60         score -= value;
61     }
62     }
63     }
64     }
65     return score;
66 }
67

```

获取棋盘上可下的位置

代码块

```

1    // 获取当前棋盘上所有可下的位置
2    std::vector<std::pair<int, int>> GetPossibleMoves()
3    {
4        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

```
1 // 递归实现最小最大算法,加上Alpha-Beta 剪枝,用于AI决策
2 int Minimax(int depth, int alpha, int beta, bool maximizingPlayer)
3 {
4     if (depth == 0)
5         return EvaluateBoard();
6
7     auto moves = GetPossibleMoves();
8     if (moves.empty())
9         return EvaluateBoard();
10
11     if (maximizingPlayer)
12     {
13         int maxEval = INT_MIN;
14         for (const auto &move : moves)
15         {
16             int x = move.first, y = move.second;
17             board[y][x] = PIECE_BLACK;
18             if (CheckWin(x, y))
19             {
20                 board[y][x] = PIECE_EMPTY;
21                 return INT_MAX;
22             }
23             int eval = Minimax(depth - 1, alpha, beta, false);
24             board[y][x] = PIECE_EMPTY;
25             maxEval = std::max(maxEval, eval);
26             alpha = std::max(alpha, eval);
27             if (beta <= alpha)
28                 break;
29         }
30         return maxEval;
31     }
32     else
33     {
34         int minEval = INT_MAX;
35         for (const auto &move : moves)
36         {
37             int x = move.first, y = move.second;
38             board[y][x] = PIECE_WHITE;
39             if (CheckWin(x, y))
40             {
41                 board[y][x] = PIECE_EMPTY;
42                 return INT_MIN;
43             }
44             int eval = Minimax(depth - 1, alpha, beta, true);
45             board[y][x] = PIECE_EMPTY;
46             minEval = std::min(minEval, eval);
47             beta = std::min(beta, eval);
```

```

48         if (beta <= alpha)
49             break;
50     }
51     return minEval;
52 }
53 }

```

AI落子逻辑

代码块

```

1  void AIPlay(bool hardMode)
2  {
3      auto moves = GetPossibleMoves();
4      if (moves.empty())
5          return;
6
7      int bestScore = (currentPlayer == PIECE_BLACK) ? INT_MIN : INT_MAX;
8      std::pair<int, int> bestMove = moves[0];
9
10     for (const auto &move : moves)
11     {
12         int x = move.first, y = move.second;
13         board[y][x] = currentPlayer;
14         if (CheckWin(x, y))
15         {
16             board[y][x] = PIECE_EMPTY;
17             bestMove = move;
18             break;
19         }
20         int score;
21         if (hardMode)
22         {
23             score = Minimax(3, INT_MIN, INT_MAX, currentPlayer ==
PIECE_WHITE);
24         }
25         else
26         {
27             score = EvaluateBoard();
28             if (currentPlayer == PIECE_WHITE)
29                 score = -score;
30         }
31         board[y][x] = PIECE_EMPTY;
32

```

```

33         if (currentPlayer == PIECE_BLACK)
34         {
35             if (score > bestScore)
36             {
37                 bestScore = score;
38                 bestMove = move;
39             }
40         }
41         else
42         {
43             if (score < bestScore)
44             {
45                 bestScore = score;
46                 bestMove = move;
47             }
48         }
49     }
50
51     int x = bestMove.first, y = bestMove.second;
52     board[y][x] = currentPlayer;
53     moveHistory.push(bestMove);
54     PlaySound(placeSound);
55
56     if (CheckWin(x, y))
57     {
58         gameState = (currentPlayer == PIECE_BLACK) ? STATE_BLACK_WIN :
STATE_WHITE_WIN;
59         if (gameMode == MODE_PVP || currentPlayer == PIECE_BLACK)
60         {
61             PlaySound(winSound);
62         }
63         else
64         {
65             PlaySound(aiWinSound);
66         }
67     }
68     else
69     {
70         currentPlayer = (currentPlayer == PIECE_BLACK) ? PIECE_WHITE :
PIECE_BLACK;
71     }
72 }

```

代码块

```
1      // 绘制游戏右侧UI,边栏部分
2      void DrawGameUI()
3      {
4          DrawRectangle(700, 0, 300, WINDOW_HEIGHT, UI_BG_COLOR);
5          const char *playerText = (currentPlayer == PIECE_BLACK) ? "Now: Black"
: "Now: White";
6          DrawText(playerText, 720, 20, 24, TEXT_COLOR);
7
8          const char *modeText = "";
9          switch (gameMode)
10         {
11             case MODE_PVP:
12                 modeText = "Pair Game";
13                 break;
14             case MODE_PVE_EASY:
15                 modeText = "AI game(Easy)";
16                 break;
17             case MODE_PVE_HARD:
18                 modeText = "AI Game(Hard)";
19                 break;
20         }
21         DrawText(modeText, 720, 60, 20, TEXT_COLOR);
22
23         // 悔棋按钮
24         Button btnUndo = CreateButton(720, 100, 150, 40, "Undo");
25         btnUndo.isHovered = CheckCollisionPointRec(GetMousePosition(),
btnUndo.bounds);
26         DrawButton(&btnUndo);
27
28         // 清空棋盘按钮
29         Button btnClear = CreateButton(720, 150, 150, 40, "Clear Board");
30         btnClear.isHovered = CheckCollisionPointRec(GetMousePosition(),
btnClear.bounds);
31         DrawButton(&btnClear);
32
33         // 返回菜单按钮
34         Button btnMenu = CreateButton(720, 200, 150, 40, "Back to Menu");
35         btnMenu.isHovered = CheckCollisionPointRec(GetMousePosition(),
btnMenu.bounds);
36         DrawButton(&btnMenu);
37
38         // 音乐控制按钮
39         Button btnMusic = CreateButton(720, 250, 150, 40, musicPlaying ? "Mute"
: "Play Music");
```

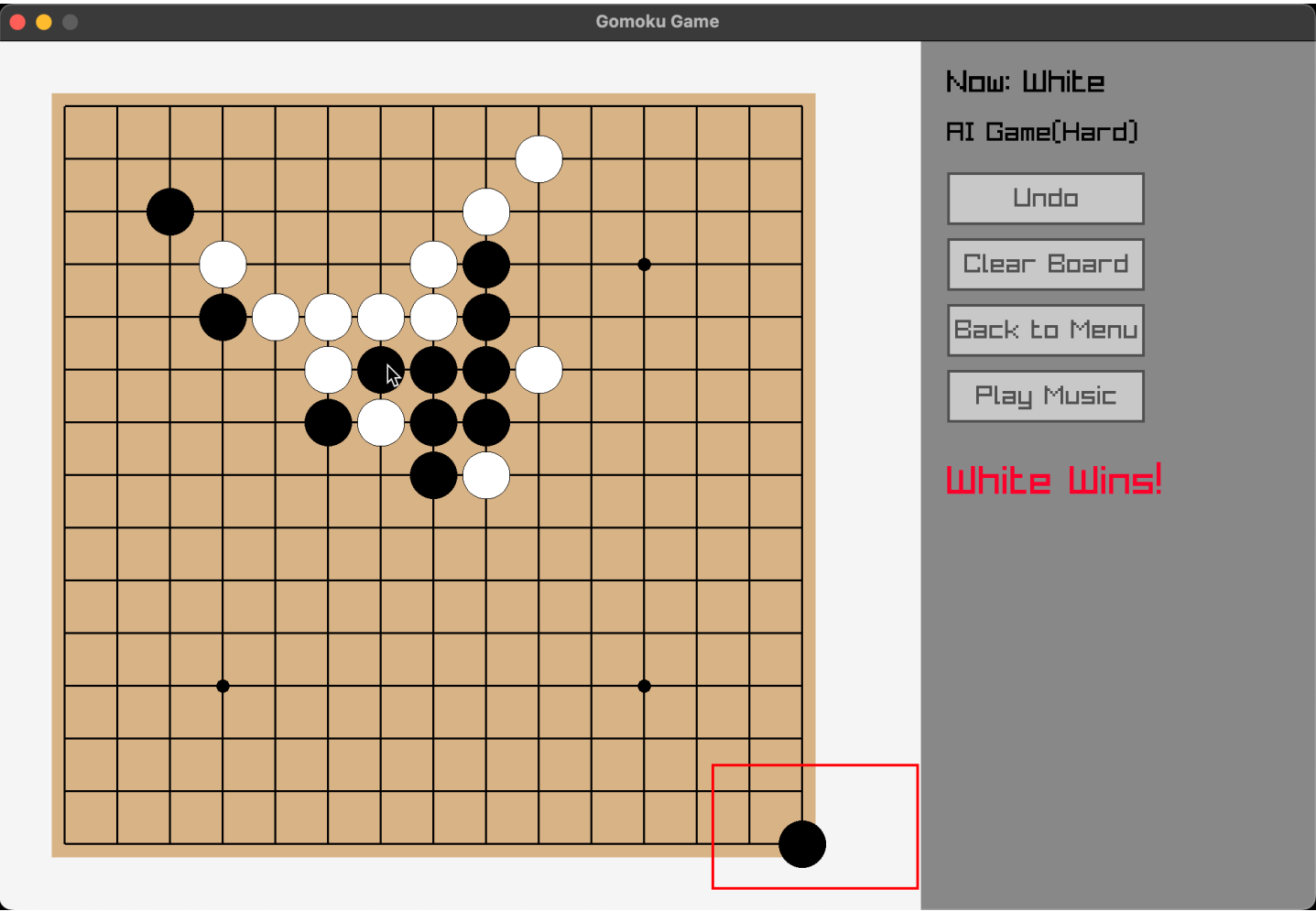
```
40         btnMusic.isHovered = CheckCollisionPointRec(GetMousePosition(),
btnMusic.bounds);
41         DrawButton(&btnMusic);
42
43         if (IsMouseButtonPressed(MOUSE_LEFT_BUTTON))
44         {
45             if (btnUndo.isHovered)
46             {
47                 UndoMove();
48             }
49             else if (btnClear.isHovered)
50             {
51                 ClearBoard();
52             }
53             else if (btnMenu.isHovered)
54             {
55                 gameState = STATE_MENU;
56             }
57             else if (btnMusic.isHovered)
58             {
59                 musicPlaying = !musicPlaying;
60                 if (musicPlaying)
61                 {
62                     PlayMusicStream(bgMusic);
63                 }
64                 else
65                 {
66                     StopMusicStream(bgMusic);
67                 }
68             }
69         }
70
71         if (gameState == STATE_BLACK_WIN)
72         {
73             DrawText("Black Wins!", 720, 320, 30, RED);
74         }
75         else if (gameState == STATE_WHITE_WIN)
76         {
77             DrawText("White Wins!", 720, 320, 30, RED);
78         }
79     }
80
81     // 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

KaiHaverz / Gomoku_Game

Q Type to search

<> Code

Issues

Pull requests

Actions

Projects

Wiki

Security

Insights

Settings

Gomoku_Game

Public

Pin

Unwatch 1

Fork 0

Star 0

master 1 Branch 0 Tags

Go to file

+<> Code

KaiHaverz

重构代码

077e9bd · yesterday 13 Commits

.vscode	delete some sound files	yesterday
Sound	delete some sound files	yesterday
src	重构代码	yesterday
README.md	重构代码	yesterday
gomoku	重构代码	yesterday

README

基于Raylib的五子棋小游戏

1 开发环境

VsCode

About

NKU2025,高级语言程序设计2-2,课程大作业.

Readme

Activity

0 stars

1 watching

0 forks

Releases

No releases published

Create a new release

Packages

No packages published

Publish your first package

Languages

C++ 100.0%

https://github.com/KaiHaverz/Gomoku_Game/tags

- 提交次数: 15

- 10995++
- 9495--



KaiHaverz

15 commits 10,995 ++ 9,495 --

#1

