五子棋程序开发日志

作者：刘继轩

2400017722

最后修改日期：2024 年 11 月 29 日

目录

1 2024 年 11 月 13 日

1.1 今日进展

今天了解了作业的具体要求，了解了使用 C++ 实现五子棋对弈程序所需要的算法基础，即 Min-Max 算法和 Alpha-Bata 剪枝优化。然后在 GitHub 上创建了仓库，方便后续的版本控制和更新。并使用 GPT 生成了 latex 模板方便后续开发日志的记录。同时今天找到了一些宝贵的学习参考资源，比如 GitHub 上基于 Javascript 语言的五子棋 AI 教程https://github.com/lihongxun945/gobang?tab=readme-ov-file 和 bilibili 上的算法教程视频https://www.bilibili.com/video/BV1v94y1r7F8/?spm_id_from=333.880.my_history.page.click&vd_source=2f0075ad419feeef529bb2dce0adc975。

1.2 本次作业的基本要求

1.2.1 五子棋详细规则

黑白双方轮流落子，黑方为先手。

在横、竖、斜方向上连成五子（连续五个棋子皆为己方）者为胜。

黑棋在行棋过程中，如果违反以下“禁手规则”会被判负。

三三禁手：黑棋在一个位置下子后，形成两个或两个以上的活三。活三是指在棋盘上有三个连续的黑子，并且两端都有空位可以继续下子形成五连珠。

四四禁手：黑棋在一个位置下子后，形成两个或两个以上的活四。活四是指在棋盘上有四个连续的黑子，并且至少有一个空位可以继续下子形成五连珠。

长连禁手：黑棋在一个位置下子后，形成六个或更多连续的黑子。

四三禁手：黑棋在一个位置下子后，同时形成一个活四和一个活三。这种情况也被视为禁手。

注意到这里的禁手规则，后续需要特定的函数实现。

1.2.2 其他要求

棋盘大小可以自定义，如果要参加 Botzone https://botzone.org.cn/ 比赛，则棋盘大小为 15*15。

1.3 算法基础介绍

Min-Max 算法：

五子棋看起来有各种各样的走法，而实际上把每一步的走法展开，就是一颗巨大的博弈树。在这个树中，从根节点为 0 开始，奇数层表示电脑可能的走法，偶数层表示玩家可能的走法。

那么我们如何才能知道哪一个分支的走法是最优的，我们就需要一个评估函数能对当前整个局势作出评估，返回一个分数。我们规定对电脑越有利，分数越大，对玩家越有利，分数越小，分数的起点是 0。

我们遍历这颗博弈树的时候就很明显知道该如何选择分支了：

电脑走棋的层我们称为 MAX 层，这一层电脑要保证自己利益最大化，那么就需要选分最高的节点。

玩家走棋的层我们称为 MIN 层，这一层玩家要保证自己的利益最大化，那么就会选分最低的节点。

而每一个节点的分数，都是由子节点决定的，因此我们对博弈树只能进行深度优先搜索而无法进行广度优先搜索。深度优先搜索用递归非常容易实现，然后主要工作其实是完成一个评估函数，这个函数需要对当前局势给出一个比较准确的评分。

alpha-beta 剪枝：即每次更新节点的数值时，查看其是否被父节点所兼容：如果父节点已经得到了合理的结果，就可以通过 break 语句进行剪枝。

1.4 明日计划

编写一些基本的函数，实现输入与输出的读取。

1.5 后续待实现的内容

胜负判断函数；禁手规则判断函数；局势评估函数；

2 2024 年 11 月 27 日

2.1 今日进展

1. 完成了棋盘初始化、显示、落子验证的实现。

2. 实现了胜负判断逻辑，通过检查横向、纵向、两种斜向的五子连珠状态，判断玩家是否获胜。

3. 实现了黑白棋的交替落子逻辑，并在有效落子后实时更新棋盘。

4. 优化了终端显示，解决了中文输出乱码问题。

2.2 成果展示

2.2.1 程序界面截图

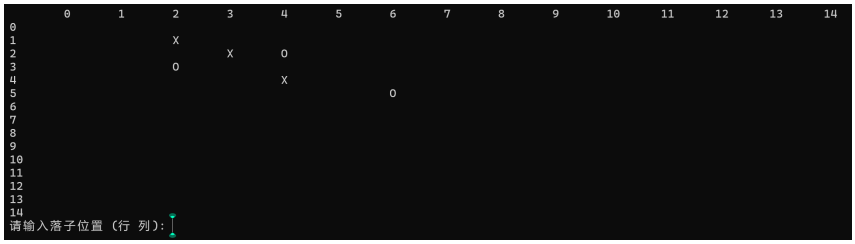

图 1: 程序运行截图

2.2.2 运行结果

程序成功运行，能够初始化指定大小的棋盘，支持黑白棋交替落子，并正确判断胜负。

2.3 问题与解决方案

2.3.1 遇到的问题

1. 在终端中显示中文提示时出现乱码。

2. 无法确定落子是否形成五子连珠，导致游戏胜负判断逻辑缺失。

2.3.2 解决方法

1. 使用 ‘SetConsoleOutputCP(65001)‘ 将控制台编码设置为 UTF-8，解决了中文乱码问题。

2. 编写了 ‘checkwin‘ 函数，通过遍历四个方向检查当前玩家是否形成五子连珠。

2.4 代码分析

2.4.1 棋盘初始化与显示

```
vector<vector<char>> initializeBoard(int size)
{
    return vector<vector<char>>(size, vector<char>(size, '␣'));
}

void displayBoard(const vector<vector<char>>& board)
{
    cout << "\t";
    for (int i = 0; i < board.size(); ++i) {
        cout << i << "\t";
    }
    cout << endl;

    for (int i = 0; i < board.size(); ++i) {
        cout << i << "\t";
        for (int j = 0; j < board[i].size(); ++j) {
            cout << board[i][j] << "\t";
        }
        cout << endl;
    }
}
```

Listing 1: 棋盘初始化与显示代码

2.4.2 胜负判断核心代码

```
bool checkwin(const vector<vector<char>>& board, int x, int y, int
   currentPlayer) {
    int directions[4][2] = {{1, 0}, {0, 1}, {1, 1}, {1, -1}};
    for (auto direction : directions) {
        int count = 1;

        for (int i = 1; i < 5; i++) {
            int nx = x + i * direction[0];
            int ny = y + i * direction[1];
            if (nx >= 0 && nx < board.size() && ny >= 0 && ny < board
               .size() &&
```

```
                board[nx][ny] == currentPlayerchar[currentPlayer]) {
                count++;
            } else {
                break;
            }
        }
        for (int i = 1; i < 5; i++) {
            int nx = x - i * direction[0];
            int ny = y - i * direction[1];
            if (nx >= 0 && nx < board.size() && ny >= 0 && ny < board
                .size() &&
                board[nx][ny] == currentPlayerchar[currentPlayer]) {
                count++;
            } else {
                break;
            }
        }
        if (count == 5) return true;
    }
    return false;
}
```

Listing 2: 五子连珠胜负判断代码

2.5 明日计划

1. 增加平局检测功能。
2. 实现黑棋禁手规则（如三三禁手、四四禁手）。
3. 为游戏添加菜单功能，支持重新开始或退出。

3 2024 年 11 月 29 日

3.1 今日进展

1. 禁手规则判断：

- 实现了五子棋的禁手规则检测，包括四种禁手类型：三三禁手、四四禁手、四三禁手和长连禁手。通过检测棋盘上的活三、活四和长连等棋型，成功防止了 AI 进行禁手操作。

2. 局面评估功能：

- 完成了初步的局面评估功能，结合禁手规则，能够根据当前棋局对每个落子进行评分，评估玩家的进攻和防守局面。

3. 菜单功能：

- 实现了一个简单的游戏菜单，允许玩家设置黑棋（X）和白棋（O）的玩家类型（人类或 AI）。通过菜单，玩家可以选择是否开始游戏或退出程序。

4. AI 随机算法：

- 为 AI 实现了一个简单的随机落子算法，使得 AI 能够在棋盘上随机选择一个空位置进行落子。

5. 输入函数设计：

- 设计并实现了一个输入函数，方便了后续 AI 模块的接入。输入函数根据玩家类型（人类或 AI）动态选择适当的输入方式，简化了 AI 和人类玩家之间的交互。

3.2 成果展示

3.2.1 程序界面截图

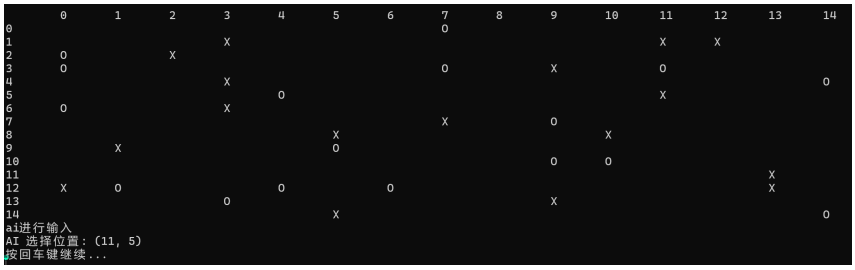

图 2: AI 自我对弈运行截图

3.2.2 运行结果

1. 禁手规则检测功能成功运行，能够在 AI 落子时避免三三禁手、四四禁手等情况。

2. 随机 AI 在每轮对局中能够随机选择空位置进行落子，且避免禁手操作。

3. 游戏菜单能够正常运行，玩家能够选择黑棋和白棋的玩家类型，并选择开始游戏或退出程序。

4. 目前，AI 和玩家在同一个棋盘上交替落子，程序能够判断胜负、平局以及禁手，确保游戏规则正常执行。

5. 输入函数已成功集成，允许根据玩家类型选择合适的输入方式，便于后续 AI 模块的接入。

3.3 问题与解决方案

3.3.1 遇到的问题

1. 禁手规则判断存在逻辑漏洞：

- 初期禁手规则检测时，发现没有正确处理一些复杂局面，尤其是四四禁手和三三禁手的检测，导致 AI 在某些情况下可能仍会选择禁手棋步。

2. AI 决策过于简单：

- 当前的 AI 只使用随机算法选择棋步，虽然可以避免禁手，但缺乏战略性，导致游戏体验较为简单，缺乏挑战性。

3. 棋盘显示与用户交互较为简单：

- 当前的程序仅依赖终端输出，缺少图形化界面，用户体验较差。

3.3.2 解决方法

1. 禁手规则优化：

- 对三三禁手和四四禁手的判断进行了优化。通过细化禁手检测函数，增加了对活三和活四的严格检查，确保 AI 在选择棋步时能够避开这些禁手规则。

2. AI 改进：

- 目前的 AI 仍然依赖随机算法，下一步将实现基于局面评估函数的 Minimax 算法，并结合 Alpha-Beta 剪枝提升 AI 的智能程度。

3. 游戏菜单优化：

- 完成了游戏菜单功能，能够实现玩家与 AI 之间的设置。接下来会考虑实现简单的图形界面，以提高用户体验，使游戏界面更加直观。

4. 输入函数集成：

- 设计并实现了一个简洁的输入函数，解决了不同玩家类型（人类或 AI）之间的输入方式问题。这个函数简化了后续 AI 模块的接入，使得系统更加灵活。

3.4 代码分析

3.3.1 禁手规则检测代码

```
bool isForbiddenMove(const vector<vector<char>>& board, int x, int y)
{
    int liveThreeCount = 0, liveFourCount = 0;
```

```
4
5       for (auto dir : directions)
6       {
7           if (isLiveThree(board, x, y, 'X', dir[0], dir[1]))
                liveThreeCount++;
8           if (isLiveFour(board, x, y, 'X', dir[0], dir[1]))
                liveFourCount++;
9       }
10
11      // 禁手规则判断
12      if (isOverline(board, x, y, 'X'))
13      {
14          cout << "长连禁手！" << endl;
15          return true;
16      }
17      if (liveThreeCount >= 2)
18      {
19          cout << "三三禁手！" << endl;
20          return true;
21      }
22      if (liveFourCount >= 2)
23      {
24          cout << "四四禁手！" << endl;
25          return true;
26      }
27      if (liveThreeCount >= 1 && liveFourCount >= 1)
28      {
29          cout << "四三禁手！" << endl;
30          return true;
31      }
32
33      return false; // 没有禁手
34  }
```

Listing 3: 禁手规则判断核心代码

3.4.2 菜单功能代码

```
1   void displayMenu(bool &blackPlayerType, bool &whitePlayerType)
2   {
```

```
    cout << "====五子棋游戏菜单====" << endl;
    cout << "1.设置玩家类型" << endl;
    cout << "2.开始游戏" << endl;
    cout << "3.退出游戏" << endl;

    int choice = -1;

    while(true)
    {
        cout << "请选择: " << endl;
        cin >> choice;
        switch(choice)
        {
            case 1:
                cout << "选择黑棋（X）玩家类型:" << endl;
                cout << "1.人类玩家" << endl;
                cout << "2.AI␣玩家" << endl;
                int blackChoice;
                cin >> blackChoice;
                blackPlayerType = (blackChoice == 1) ? false : true;
                cout << "选择白棋（O）玩家类型:" << endl;
                cout << "1.人类玩家" << endl;
                cout << "2.AI␣玩家" << endl;
                int whiteChoice;
                cin >> whiteChoice;
                whitePlayerType = (whiteChoice == 1) ? false : true;
                break;
            case 2:
                cout << "开始游戏..." << endl;
                return; // 返回，开始游戏
            case 3:
                cout << "退出游戏..." << endl;
                exit(0); // 退出程序
            default:
                cout << "请重新选择!" << endl;
        }
    }
}
```

Listing 4: 游戏菜单功能代码

3.4.3 输入函数代码

```
void processInput(const string& playerType, const vector<vector<char
    >>& board, char currentPlayer) {
    if (playerType == "human") {
        manualInput(board);
    } else if (playerType == "ai") {
        aiInput(board);
    }
}

void manualInput(const vector<vector<char>>& board) {
    int x, y;
    cout << "请输入落子位置␣(行␣列):␣";
    cin >> x >> y;
    // 检查位置有效性
}

void aiInput(const vector<vector<char>>& board) {
    // 简单随机AI输入
    srand(time(0)); // 随机种子
    int x = rand() % board.size();
    int y = rand() % board.size();
    cout << "AI␣选择位置:␣(" << x << ",␣" << y << ")" << endl;
}
```

Listing 5: 输入函数设计

3.5 明日计划

1. 完善 Minimax 算法，结合 Alpha-Beta 剪枝提升 AI 智能，减少计算时间。
2. 实现 AI 基于评估函数的决策逻辑，替代随机算法，提升游戏体验。
3. 开始实现简单的图形化界面，展示棋盘、玩家及 AI 的落子。
4. 优化禁手检测，确保所有禁手类型都能被准确识别并避免。

参考文献

[1] 作者, 书名, 出版社, 出版年份.

[2] 在线资源标题, https://example.com