華中科技大學

课 程 实 验 报 告

课程名称：

基于 $\alpha-\beta$ 剪枝算法的五子棋游戏

专业班级 CS2201

学　　号 U202215357

姓　　名 王文涛

指导教师 冯琪

报告日期 2024 年 1 月 2 日

计算机科学与技术学院

目　　录

1 摘要 1

2 问题描述 2

3 $\alpha-\beta$ 剪枝算法 3

3.1 算法原理 3

4 五子棋原理 7

5 编码实现 9

5.1 五子棋对战方面 9

5.2 $\alpha-\beta$ 剪枝实现 11

6 小结及展望 17

1 摘要

这次实验我选择了基于 $\alpha-\beta$ 剪枝算法的五子棋游戏。一方面是因为我对于做游戏比较感兴趣，另一方面我认为 $\alpha-\beta$ 剪枝算法比较使用，在很多地方都能用上。我是用了 easyX 图形库在 VS2022 上来实现图形化界面，使用 C++ 来编程。来实现我本来打算做一个完整的图形页面出来。但是由于我没有规划好时间，导致最后只做完了一部分，有些功能来不及实现。

2 问题描述

任务包括两个部分

1. 编写五子棋游戏程序，支持人机对战。这一部分与算法无关，主要是制作游戏框架，提供平台与操作菜单。维护 AI 类，man 类和 chess 类，实现 AI 和玩家的走棋函数，和对棋局的判断函数，在一个死循环中 AI 和玩家依次走棋，有一方胜利或平局时时退出循环。
2. 编程实现 $\alpha-\beta$ 剪枝算法，作为机器方的下棋算法。对棋盘新增棋子的所有可能形成的状态空间进行 $\alpha-\beta$ 剪枝，以求在较短时间内得到最优（棋盘得分最高）的落子位置。

3 $\alpha-\beta$ 剪枝算法

五子棋的 AI 使用 $\alpha-\beta$ 剪枝算法实现。

3.1 算法原理

3.1.1 MINMAX 算法

$\alpha-\beta$ 剪枝算法是对 MINMAX 算法的优化。假如有如上图的博弈树，设先手为 A，后手为 B；则 A 为 max 局面，B 为 min 局面。图 3-1 中 A 一开始有 2 种走法 (w2 和 w3，w 表示结点记号)，它走 w2 还是 w3 取决于 w2 和 w3 的估价函数值 f()，因为 A 是 max 局面，所以它会取 f(w2) 和 f(w3) 中大的那个，f(x) 通常是以递归的方式对博弈树进行搜索，可以设定叶子结点局面的估价值。如上图的搜索过程为 w1 –> w2 –> w4，然后回溯到 w1 –> w2 得到 f(w2) = 3，接着 w1 –> w2 –> w5 得到 f'(w2) = 1，因为 w2 在第二层，是 min 局面，所以它会选择得到的结果中小的那个，即用 f'(w2) 替代 f(w2)，即 f(w2) = 1，接着 w1 –> w2 –> w6 得到 f'(w2) = 6 > f(w2)，直接忽略。因此如果 A 往 w2 走的话将会得到一个估价值为 f(w2) = 1 的局面；类似地，如果往 w3 走的话将会得到一个估价值为 f(w3) = -3 的局面。而 A 是 max 局面，所以它会选择估价值大的走法，f(w2) = 1 > f(w3) = -3，因此它下一步走 w2。

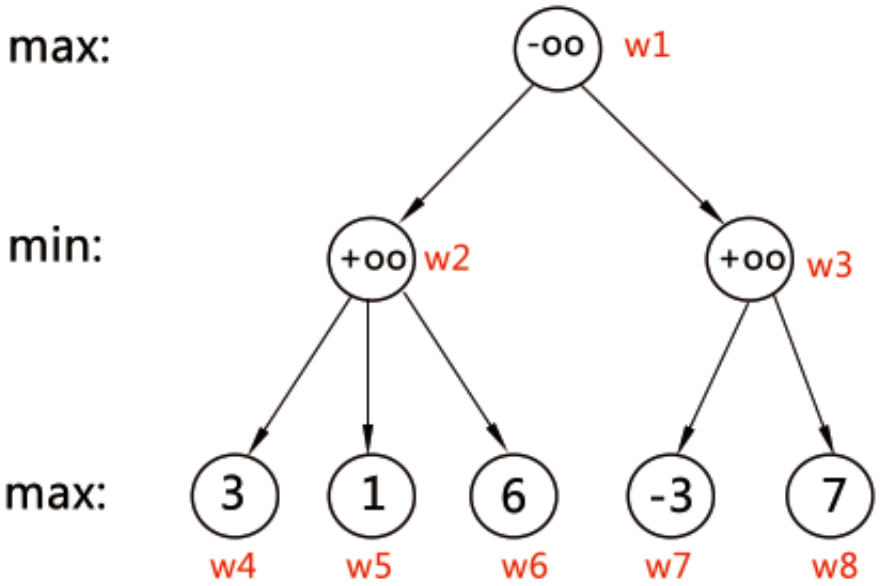

图 3-1　MINMAX 算法原理图

3.1.2　$\alpha-\beta$ 剪枝算法

alpha 初始值为 -oo，beta 初始值为 +oo；w1 在得到 f(w1) = 6 的整个过程中，alpha 和 beta 是这样变化的：w1 –> w2 – > w5 得到 f'(w2) = 6 < f(w2) = +oo，即修改 f(w2) = f'(w2) = 6；同时修改 beta = 6 (因为 w2 的子局面是否需要剪枝依赖于 w2 的估价值 f()，而与 alpha 无关，故不需修改 alpha)。在正常搜索完 w1 –> w2 –> w6 后，w2 层把 beta = 6 返回给上一层 w1 的 alpha，即 w1 层的 alpha = 6，beta = +oo。

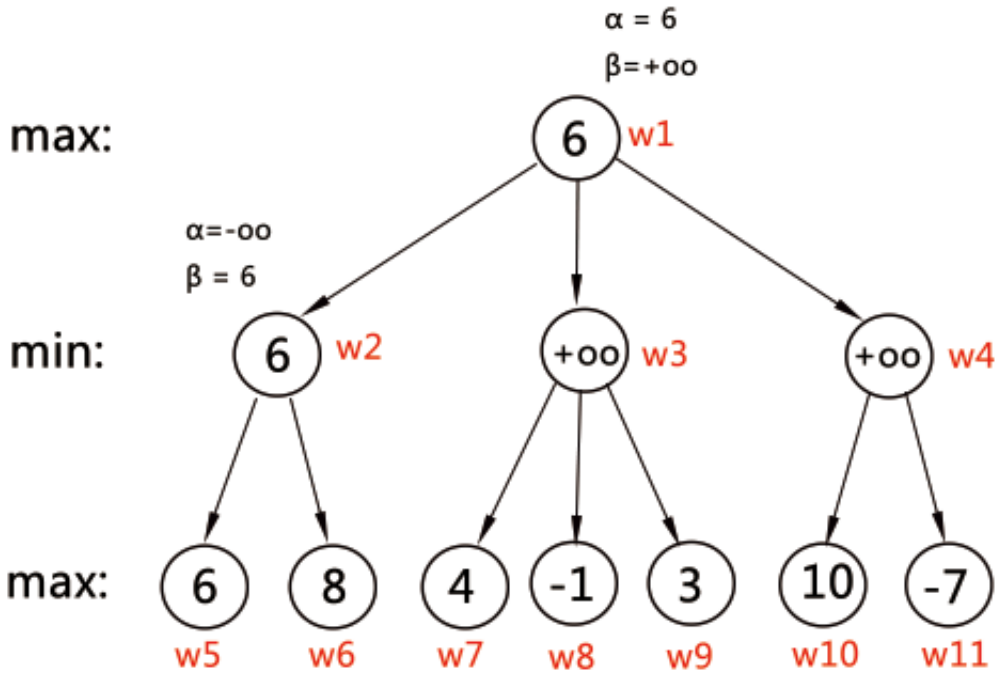

图 3-2　$\alpha-\beta$ 剪枝算法原理图

接下来在 w3 里搜索就可以体现 alpha-beta 剪枝的效果；A 在选择 w3 走的时候同时把所得的 alpha 和 beta 传递下去，在经过 w1 –> w3 –> w7 得到 f(w3) = 4 (同时使 beta = 4) 后，首先进行判断：如果 alpha > beta，则直接返回 beta = 4，没有必要再搜索 w8 和 w9。这是因为这个 alpha 是 w1 在走 w2 这条路时得到的一个估价值 f(w1)，而 w3 是 min 局面，它会选择子局面 w7, w8, w9 中 f() 的值小的作为 f(w3)，所以 w3 在得到 f(w3) = 4 后如果继续搜索 w8, w9，只会得到更小的值；w1 是 max 局面，它的 f() 要修改的条件是找到估价值比它更大的子局面，而 w1 目前已知的估价值 f(w1) = 6 比 f(w3) 要大，所以无论 w3 再怎么继续搜索下去，w1 都不会选 w3 作为下一步，所以没有必要搜索下去。这样就剪掉了 w8, w9 这两个分支，直接跳出 w3 进入 w4 继续搜索。另外一种情形也是类似的道理，这样就实现了有效的剪枝优化。

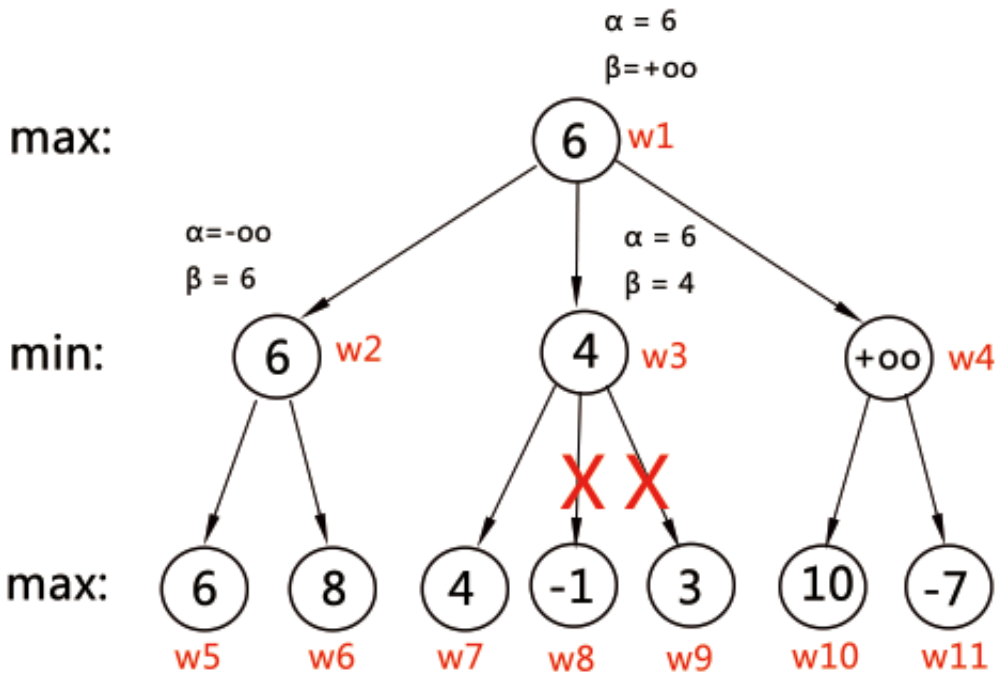

图 3-3　$\alpha-\beta$ 剪枝算法原理图

4 五子棋原理

根据百度百科的介绍，五子棋起源于中国，是一种两人对弈的纯策略型棋类游戏。双方分别使用黑白两色的棋子，下在棋盘直线与横线的交叉点上，先形成五子连珠者获胜。正如前面介绍 $\alpha-\beta$ 剪枝算法时所说，我们需要一个评估函数来对棋盘上不同的棋形进行评分，来作为 AI 下棋的依据。我们首先要了解五子棋的一些术语。

- 阳线：直线，棋盘上可见的横纵直线。
- 阴线：斜线，由交叉点构成的与阳线成 45° 夹角的隐形斜线。
- 长连：五枚以上同色棋子在一条阳线或阴线上相邻成一排。
- 五连：只有五枚同色棋子在一条阳线或阴线上相邻成一排。
- 活四：有两个点可以成五的四。
- 冲四：只有一个点可以成五的四。
- 死四：不能成五的四。
- 活三：再走一着可以形成活四的三。
- 连活三：连续、中间不隔空点的活三，即同色棋子在一条阳线或阴线上相邻成一排的活三。简称“连三”。
- 跳活三：中间隔有一个空点的活三。简称“跳三”。
- 眠三：再走一着可以形成冲四的三。
- 死三：不能成五的三。

说到这里，其实对五子棋稍有一些了解的人，都应该能发现先手执黑的胜率异常的大，但实际上，如果没有禁手，先手执黑其实是必胜的，大家有兴趣的话可以去网上搜索一番，什么花月蒲月之类的东西。所以我们不能避开的就是聊聊什么是禁手。整个对局过程中黑方有禁手，白方无禁手。黑方禁手有三三禁手、四四禁手和长连禁手三种。

- 三三禁手：黑方一子落下同时形成两个或两个以上的活三，此步为三三禁手。
- 四四禁手：黑方一子落下同时形成两个或两个以上的四，活四、冲四、嵌五之四，包括在此四之内，此步为四四禁手。
- 长连禁手：黑方一子落下形成连续六子或六子以上相连，此步为长连禁手。

实际上由于时间有限，我省略了黑棋禁手的部分。

5 编码实现

5.1 五子棋对战方面

我使用 C++ 创建了 game,chess,AI,man 四个类。

5.1.1 game 类

game 类用来创建游戏循环。

```
class game
{
    Man* man;
    AI* ai;
    Chess* chess;
public:
    void play();//控制游戏循环
    game(Man* man, AI* ai, Chess* chess);
};
```

5.1.2 chess 类

chess 类用来保存棋局。

```
class Chess
{
    //加载图片
    IMAGE blackChess;
    IMAGE whiteChess;
    IMAGE target;
    //棋盘属性
    const int margin_x;//边距
    const int margin_y;
    const int squareWidth;//格子宽度
```

```
        void updateMap(const ChessPosition& pos);//更新棋盘
    public:
        Chess(int margin_x, int margin_y, int squareWidth);
        void init();
        bool clickBoard(const int &x, const int &y, ChessPosition &pos
        void chessDown(const ChessPosition& pos, const int &kind);//执
        void putTarget(const ChessPosition& pos);//渲染放置提示
        int getsize();//获取棋盘大小
        bool gameOver();//游戏结束
        int size();
        int mapData(const int& x, const int& y) const;
        static const int boardSize = 15;
        int chessMap[15][15];//棋盘数组
        int player;//表示现在到谁下棋
    };
```

5.1.3 man 类

man 类表示玩家走棋

```
    class Man
    {
        Chess* chess;
    public:
        void init(Chess* chess);
        void go(int player);
    };
```

5.1.4 AI 类

AI 类控制 AI 走棋

```
    class AI
{
```

```
    Chess* chess;
    int map[15][15];
public:
    void init(Chess* chess);
    bool isValid(int x, int y);//判断落子位置是否合法
    void go(int player);//AI走棋
    int heuristic(int player);//评估整个棋盘
    int evaluate(int x, int y, int color);//评估棋形
    int AlphaBeta(int depth, int alpha, int beta,int player);//$\a
};
```

5.2 $\alpha-\beta$ 剪枝实现

```
int AI::AlphaBeta(int depth, int alpha, int beta,int player)
{
    vector<Point> p;
    //游戏结束或递归到达边界
    if (depth == 0 || chess->gameOver()) {
        return heuristic(player);//返回棋局分数
    }
    // 生成候选步
    for (int i = 0; i < chess->boardSize; i++)
        for (int j = 0; j < chess->boardSize; j++) {
            if (chess->chessMap[i][j])
                p.push_back(Point(i, j));
        }
    sort(p.begin(), p.end());//对候选步进行排序，提高剪枝效率
    //遍历候选步
    for (auto i : p) {
        chess->chessMap[i.x][i.y] = player;
        int value= -AlphaBeta(depth - 1, -beta, -alpha, -player);
```

```
            chess->chessMap[i.x][i.y] = NONE;
            if (value > alpha) {
                // alpha + beta剪枝点
                if (value >= beta) {
                    return beta;
                }
                alpha = value;
            }
        }
        return alpha;
    }
```

剪枝中评估函数非常重要。我使用 01 字符串来表示不同的棋形和对应的分数，然后在棋盘中中的每个子周围寻找存在的棋形，进行相加，得到对应位置的分数。

```
    int AI::evaluate(int x, int y, int color)
    {
        //表示不同的局面的字符串对应的分数
        const int CHENG_5_SCORE = 5000000;//成五
        const int HUO_4_SCORE = 100000;//活四
        const int CHONG_4_SCORE = 10000;//冲四
        const int DAN_HUO_3_SCORE = 8000;//单活三
        const int TIAO_HUO_3_SCORE = 7000;//跳活三
        const int MIAN_3_SCORE = 500;//眠三
        const int HUO_2_SCORE = 50;//活二
        const int MIAN_2_SCORE = 10;//眠二
        //表示不同的局面的字符串
        string CHENG_5_STRING = "11111";
        string HUO_4_STRING = "011110";
        string CHONG_4_STRING_1_1 = "01111-";
        string CHONG_4_STRING_1_2 = "-11110";
        string CHONG_4_STRING_2_1 = "10111";
```

```
string CHONG_4_STRING_2_2 = "11101";
string CHONG_4_STRING_3 = "11011";
string DAN_HUO_3_STRING = "01110";
string TIAO_HUO_3_STRING_1_1 = "1011";
string TIAO_HUO_3_STRING_1_2 = "1101";
string MIAN_3_1_1 = "00111-";
string MIAN_3_1_2 = "-11100";
string MIAN_3_2_1 = "01011-";
string MIAN_3_2_2 = "-11010";
string MIAN_3_3_1 = "01101-";
string MIAN_3_3_2 = "-10110";
string MIAN_3_4_1 = "10011";
string MIAN_3_4_2 = "11001";
string MIAN_3_5 = "10101";
string MIAN_3_6 = "-01110-";
string HUO_2_STRING_1 = "001100";
string HUO_2_STRING_2 = "01010";
string HUO_2_STRING_3 = "1001";
string MIAN_2_1_1 = "00011-";
string MIAN_2_1_2 = "-11000";
string MIAN_2_2_1 = "00101-";
string MIAN_2_2_2 = "-10100";
string MIAN_2_3_1 = "01001-";
string MIAN_2_3_2 = "-10010";
string MIAN_2_4 = "10001";
int score = 0; // 分数
int weight = PosValue[x][y]; // 权重，距离中间越近，权重越高，
// 分别构造4个方向的局面的字符串表示
for (int dir = 0; dir < 4; dir++) {
    string s = "";
    // 计算该方向上的起始点坐标
```

```
int rBegin = x + DIRECTION[dir][0] * 4;
int cBegin = y + DIRECTION[dir][1] * 4;
// 坐标递增的方向
int rDir = -DIRECTION[dir][0];
int cDir = -DIRECTION[dir][1];
// 计算该方向上的终止点坐标
int rEnd = x + rDir * 4;
int cEnd = y + cDir * 4;
// 当行列没到终点的时候（表示没有收集齐9个点），循环
int r = rBegin;
int c = cBegin;
while (r != rEnd || c != cEnd) {
    // 如果这个点没有超过棋盘范围，是自己颜色就记为1，是空
    if (isValid(r, c))
        if (chess->chessMap[r][c] == color) s += "1";
        else if (chess->chessMap[r][c] == NONE) s += "0";
        else s += "-";
    else
        s += "#";
    r += rDir;
    c += cDir;
}
// 如果构建出来的字符串中包含“成五”的子串，加上其分数
if (s.find(CHENG_5_STRING) != string::npos)
    score += CHENG_5_SCORE;
// 如果包含“活四”的子串，加上其分数
if (s.find(HUO_4_STRING) != string::npos)
    score += HUO_4_SCORE;
// “冲四”不止一种情况，如果包含任意一个子串，加上其分数，
if (s.find(CHONG_4_STRING_1_1) != string::npos || s.find(C
    score += CHONG_4_SCORE;
```

```
        if (s.find(DAN_HUO_3_STRING) != string::npos)
            score += DAN_HUO_3_SCORE;
        if (s.find(TIAO_HUO_3_STRING_1_1) != string::npos
            || s.find(TIAO_HUO_3_STRING_1_2) != string::npos)
            score += TIAO_HUO_3_SCORE;
        if (s.find(MIAN_3_1_1) != string::npos
            || s.find(MIAN_3_1_2) != string::npos
            || s.find(MIAN_3_2_1) != string::npos
            || s.find(MIAN_3_2_2) != string::npos
            || s.find(MIAN_3_3_1) != string::npos
            || s.find(MIAN_3_3_2) != string::npos
            || s.find(MIAN_3_4_1) != string::npos
            || s.find(MIAN_3_4_2) != string::npos
            || s.find(MIAN_3_5) != string::npos
            || s.find(MIAN_3_6) != string::npos)
            score += MIAN_3_SCORE;
        if (s.find(HUO_2_STRING_1) != string::npos
            || s.find(HUO_2_STRING_2) != string::npos
            || s.find(HUO_2_STRING_3) != string::npos)
            score += HUO_2_SCORE;
        if (s.find(MIAN_2_1_1) != string::npos
            || s.find(MIAN_2_1_2) != string::npos
            || s.find(MIAN_2_2_1) != string::npos
            || s.find(MIAN_2_2_2) != string::npos
            || s.find(MIAN_2_3_1) != string::npos
            || s.find(MIAN_2_3_2) != string::npos
            || s.find(MIAN_2_4) != string::npos)
            score += MIAN_2_SCORE;
        // 四个方向的分数都加起来，乘上权重
        return score * weight;
    }
```

```
}
```

6 小结及展望

在这次实验中我完成了基于 $\alpha-\beta$ 剪枝算法的五子棋游戏的基本框架。下棋时会在离鼠标最近，且没有棋子的点上显示放置提示，如图 6-5 所示。

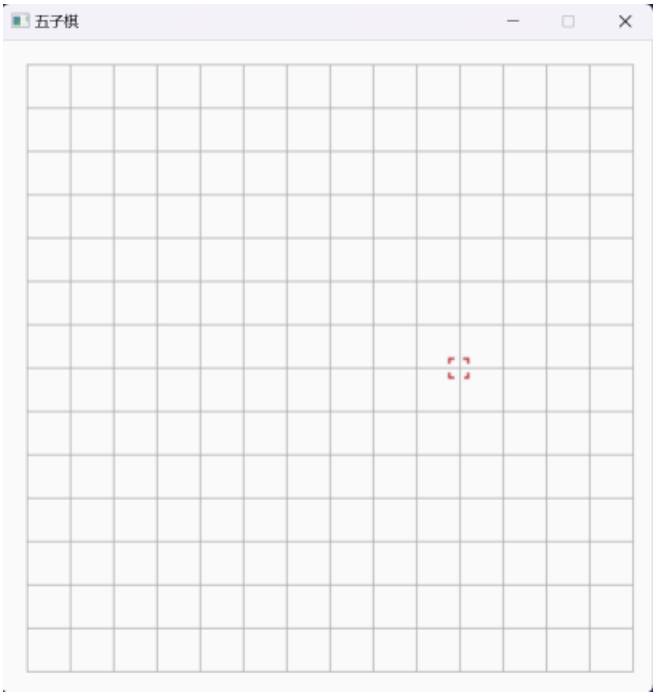

图 6-1 位置显示

但是还有很多功能没有实现。比如说黑棋禁手，选择黑子白子，悔棋等功能，接下来的空余时间里我会进一步完善，争取能与网上的一些五子棋对战平台相似。