人工智能实验九

18340146 计算机科学与技术 宋渝杰

任务：

实现 N*N 的五子棋的人机对战，要求：

- N >= 11，横排、竖排、对角线均可连线；
- 要求使用 alpha-beta 剪枝；
- 搜索深度和评价函数不限，自己设计。在报告中说明清楚自己的评价函数及搜索策略；
- 实验结果要求展示至少连续三个回合（人和机器各落子一次指一回合）的棋局分布情况，并输出每步落子的得分。

算法原理：

博弈树：一种特殊的有根树，表示两名游戏参与者之间的一场博弈，他们交替行动，试图获胜。博弈树的每一个节点存有当前游戏的信息（本实验中存有五子棋的棋盘现状）；每一条有向边可以理解为一个玩家的某种落子行为，将原棋盘现状转移至落子后的棋盘现状；每一层的所有边是同一个玩家的所有可能的决策，相邻层为不同玩家的决策，每一层的所有节点也是同一个玩家所有可能的决策带来的所有可能的棋盘现状，相邻层的节点也为不同玩家决策后的棋盘现状。

Minimax 搜索：在零和博弈中，max 玩家希望最大化自己的收益，min 玩家相对应的希望最小化对手的收益，因此在 max 玩家进行决策的时候（博弈树来到 max 节点的时候），它将会搜索所有子节点的收益，然后从中取最大值，作为自己本步骤的收益（博弈树该 max 节点的评分）；同样地，在 min 玩家进行决策的时候（博弈树来到 min 节点的时候），它也会搜索所有子节点的收益，然后从中取最小值，作为自己本步骤的收益（博弈树该 min 节点的评分）

因此，在博弈树进行 DFS 的时候，对于 max 节点，该节点的搜索结果返回所有子节点的评分的最大值，对于 min 节点，该节点的搜索结果返回所有子节点的评分的最小值

alpha-beta 剪枝：因为博弈双方都是采取最优策略，因此在整个博弈树的 Minimax 搜索中，有许多节点（棋盘现状）是不可能达到的，因此可以对这些节点采取剪枝策略，具体操作如下：

- 对于 max 节点，设 beta 是该节点被遍历过的兄弟节点中的最低值，alpha 是该节点被遍历过的子节点中的最高值，当 alpha 大于等于 beta 的时候，就可以停止遍历该节点的子节点了

  原理：前面的 min 节点一定不会来到该 max 节点，因为 min 节点一定会优先选择之前遍历过的值更小的兄弟节点

- 对于 min 节点，设 alpha 是该节点被遍历过的兄弟节点中的最高值，beta 是该节点被遍历过的子节点中的最低值，当 beta 小于等于 alpha 的时候，就可以停止遍历该节点的子节点了

  原理：前面的 max 节点一定不会来到该 min 节点，因为 max 节点一定会优先选择之前遍历过的值更大的兄弟节点

评价函数：对于某些大型博弈游戏，从时间和空间限制来看，我们无法直接遍历到最终判断出胜负的情况（博弈树的深度和宽度过大），因此一般采取深度限制搜索，且搜索到限度值时，对于该棋盘现状可以通过合理的评价函数进行当前局势的评分判断。

而对于本次实验的五子棋来说，可以通过判断某些特定的棋形（包括横向、纵向、两种斜向）来判断局势得分：（下表 'x' 表示先手棋子，'#' 表示边界或敌方棋子，'_' 表示空位）

专业术语	棋形	评分
活二	_xx_	1e2
活三	_xxx_	1e3
活四	_xxxx_	1e4
五连	xxxxx	1e6（正无穷）
眠二	_xx#	1e1
眠三	_xxx#	1e2
冲四	_xxxx#	1e3
死二	#xx#	0
死三	#xxx#	0
死四	#xxxx#	0

当棋子换为后手棋子 'o' 时，评分取反即可

因此棋盘现状的评分可以通过对 'x' 的评分减去对 'o' 的评分得到

搜索深度：本次实验采取棋盘大小 N = 18，深度 deep = 2，原因如下：

- 棋盘大小 N = 11 时棋盘过小，过于容易和棋（虽然可以支持 deep = 4，一次搜索时间约 1s~3s）
- 扩展棋盘后，深度为 4（即 AI 下 2 步玩家下 2 步）时搜索时间过长（即使加入了 alpha-beta 剪枝和其它的加速方法，AI 下一步依然超过 10s），游戏体验过差
- 深度为 3（即 AI 下 2 步玩家下 1 步）时，测试发现玩家后手时 AI 表现过差（具体在 AI 选择连自己的活三，而不去拦截玩家的活三，AI 必输）

因此采取棋盘大小 N = 18，深度 deep = 2 的方案，可以控制搜索时间为 1s 左右

伪代码/流程图：

alpha-beta 剪枝：

```
def AlphaBata(n, player, alpha, beta, deep):
    if deep == limit or n is leaf: # 深度限制or叶节点（即游戏结束）
        return V(n) # 估值函数
    if player == MAX: # max节点
        for c in n.childs: # 遍历子节点
            alpha = max(alpha, AlphaBata(c, MIN, alpha, beta, deep+1)) # 取max
            if alpha >= beta: # 剪枝
                return alpha
    else: # min节点
        for c in n.childs: # 遍历子节点
            beta = min(beta, AlphaBata(c, MAX, alpha, beta, deep+1)) # 取min
            if beta <= alpha: # 剪枝
                return beta
```

代码展示：

下面仅展示关键代码，全部代码请移步 18340146_songyujie_lab9.cpp 文件

估值函数：

```
char a[len+1][len+1];
int vis[len+1][len+1][4];

int judge(char c) { // 当前局势评分。 c：棋子类型
    int num = 0, count, jud;
    memset(vis, 0, sizeof(vis));
    for (int i=1; i<=len; i++) {
        for (int j=1; j<=len; j++) {
            if (a[i][j] == c) {
                if (vis[i][j][0] == 0) { // 横向判断
                    count = 1; jud = 0; vis[i][j][0] = 1;
                    if (j == 1 or (a[i][j-1] != ' ' and a[i][j-1] != c)) jud++;
// 左边有阻拦
                    for (int k=1; k<5; k++) {
                        if (a[i][j+k] == c) { count++; vis[i][j+k][0] = 1; } //
横向连续同类棋子
                        else if (a[i][j+k] == ' ') break; // 横向空位
                        else { jud++; break; } // 横向有其他棋子阻拦
                        if (j+k == len) { jud++; break; } // 右边到了边界
                    }
                    if (count >= 5) num += 1e6; // 胜利，取1e6近似无穷~
                    else if (count > 1) {
                        if (jud == 0) num += pow(10, count); // 活二/三/四
                        else if (jud == 1) num += pow(10, count-1); // 眠二/三/冲
四
                    }
                }
                if (vis[i][j][1] == 0) { // 纵向判断
                    count = 1; jud = 0; vis[i][j][1] = 1;
                    if (i == 1 or (a[i-1][j] != ' ' and a[i-1][j] != c)) jud++;
// 上面有阻拦
                    for (int k=1; k<5; k++) {
                        if (a[i+k][j] == c) { count++; vis[i+k][j][1] = 1; } //
纵向连续同类棋子
                        else if (a[i+k][j] == ' ') break; // 纵向空位
                        else { jud++; break; } // 纵向有其他棋子阻拦
                        if (i+k == len) { jud++; break; } // 下面到了边界
                    }
                    if (count >= 5) num += 1e6; // 胜利，取1e6近似无穷~
                    else if (count > 1) {
                        if (jud == 0) num += pow(10, count); // 活二/三/四
                        else if (jud == 1) num += pow(10, count-1); // 眠二/三/冲
四
                    }
                }
                if (vis[i][j][2] == 0) { // 左上-右下斜线
                    count = 1; jud = 0; vis[i][j][2] = 1;
                    if (i == 1 or j == 1 or (a[i-1][j-1] != ' ' and a[i-1][j-1]
!= c)) jud++; // 左上有阻拦
                    for (int k=1; k<5; k++) {
```

```
                        if (a[i+k][j+k] == c) { count++; vis[i+k][j+k][2] = 1; }
// 右下连续同类棋子
                        else if (a[i+k][j+k] == ' ') break; // 右下空位
                        else { jud++; break; } // 右下有其他棋子阻拦
                        if (i+k == len or j+k == len) { jud++; break; } // 右下到
了边界
                    }
                    if (count >= 5) num += 1e6; // 胜利，取1e6近似无穷~
                    else if (count > 1) {
                        if (jud == 0) num += pow(10, count); // 活二/三/四
                        else if (jud == 1) num += pow(10, count-1); // 眠二/三/冲
四
                    }
                }
                if (vis[i][j][3] == 0) { // 右上-左下斜线
                    count = 1; jud = 0; vis[i][j][3] = 1;
                    if (i == 1 or j == len or (a[i-1][j+1] != ' ' and a[i-1]
[j+1] != c)) jud++; // 右上有阻拦
                    for (int k=1; k<5; k++) {
                        if (a[i+k][j-k] == c) { count++; vis[i+k][j-k][3] = 1; }
// 左下连续同类棋子
                        else if (a[i+k][j-k] == ' ') break; // 左下空位
                        else { jud++; break; } // 左下有其他棋子阻拦
                        if (i+k == len or j-k == 1) { jud++; break; } // 左下到了
边界
                    }
                    if (count >= 5) num += 1e6; // 胜利，取1e6近似无穷~
                    else if (count > 1) {
                        if (jud == 0) num += pow(10, count); // 活二/三/四
                        else if (jud == 1) num += pow(10, count-1); // 眠二/三/冲
四
                    }
                }
            }
        }
    }
    return num;
}
```

Minimax（通过 alpha-beta 剪枝）：

```
int x[deep], y[deep], ma[deep+1], mi[deep+1];
int vis[len+1][len+1][4];

int minimax(char c,int l) {
    int t = judge('x')-judge('o'); // 判断当前局势
    if (t > 1e5 or t < -1e5 or l == deep) return t; // 已经胜利/达到深度，直接返回
    int ans = (c == 'x' ? -1e9 : 1e9);
    ma[l+1] = -1e9; mi[l+1] = 1e9; // 初始化遍历过的兄弟节点中的最高/最低值
    int vis[len+1][len+1];
    memset(vis, 0, sizeof(vis));
    for (int i=5; i<=len-4; i++) { // 先搜索棋盘中部
        for (int j=5; j<=len-4; j++) {
            if (a[i][j] == ' ' and vis[i][j] == 0) {
                vis[i][j] = 1;
                a[i][j] = c;
                if (c == 'x') { // max
```

```
                    int sum = minimax('o', l+1);
                    if (sum > ans) { // 子节点中最高值
                        ans = sum; x[l] = i; y[l] = j;
                    }
                    if (sum >= mi[l]) { // a >= b，剪枝
                        a[i][j] = ' ';
                        return sum;
                    }
                }
                else { // min
                    int sum = minimax('x', l+1);
                    if (sum < ans) { // 子节点中最低值
                        ans = sum; x[l] = i; y[l] = j;
                    }
                    if (sum <= ma[l]) { // b <= a，剪枝
                        a[i][j] = ' ';
                        return sum;
                    }
                }
                a[i][j] = ' '; // 回溯
            }
        }
    }
    for (int i=1; i<=len; i++) { // 再搜索外部，下面代码完全相同~
        for (int j=1; j<=len; j++) {
            if (a[i][j] == ' ' and vis[i][j] == 0) {
                vis[i][j] = 1;
                a[i][j] = c;
                if (c == 'x') { // max
                    int sum = minimax('o', l+1);
                    if (sum > ans) { // 子节点中最高值
                        ans = sum; x[l] = i; y[l] = j;
                    }
                    if (sum >= mi[l]) { // a >= b，剪枝
                        a[i][j] = ' ';
                        return sum;
                    }
                }
                else { // min
                    int sum = minimax('x', l+1);
                    if (sum < ans) { // 子节点中最低值
                        ans = sum; x[l] = i; y[l] = j;
                    }
                    if (sum <= ma[l]) { // b <= a，剪枝
                        a[i][j] = ' ';
                        return sum;
                    }
                }
                a[i][j] = ' '; // 回溯
            }
        }
    }
    mi[l] = min(mi[l], ans); // 更新遍历过的兄弟节点最高/最低值
    ma[l] = max(ma[l], ans);
    return ans;
}
```

输出棋盘：

```
void print() { // 输出棋盘
    for (int i=1; i<=len; i++) cout << "----";
    cout << "-" << endl;
    for (int i=1; i<=len; i++) {
        for (int j=1; j<=len; j++) cout << "| " << a[i][j] << " ";
        cout << "| " << len-i+1 << endl;
        for (int j=1; j<=len; j++) cout << "----";
        cout << "-" << endl;
    }
    for (int i=1; i<=len; i++) {
        if (i < 10) cout << "  " << i << " ";
        else cout << " " << i << " ";
    }
    cout << endl << endl;
}
```

先后手：

```
    while (--t) { // t = len*len-6，留几个空位提前和棋，防止中途无处下棋
        if (m == '1') { // 先手
            print(); // 输出棋盘
            int ans = judge('x')-judge('o'); // 计算当前局势
            if (-ans >= 1e5) { // 输了~
                cout << "You lose >_<" << endl;
                break;
            }
            cout << "x score: " << ans << "     o score: " << -ans << endl; // 输
出分数
            if (x[0] != 0 or y[0] != 0) cout << endl << "AI's choice:  " << y[0]
<< " " << len-x[0]+1 << endl; // 输出AI上一步的决策
            cout << endl << "Your choice:  ";
            cin >> yourx >> youry;
            while (a[len-youry+1][yourx] != ' ' or len-youry+1 > len or len-
youry+1 < 1 or youry > len or youry < 1) { // 判断输入是否合理
                cout << "You can't take this place! Choose another one:  ";
                cin >> yourx >> youry;
            }
            a[len-youry+1][yourx] = c; // 注意是二维坐标，不是矩阵坐标
            system("cls"); // 清屏
            print(); // 输出棋盘
            c = (c == 'x' ? 'o' : 'x'); // 下一步的棋子
            ans = judge('x')-judge('o'); // 计算当前局势
            if (ans >= 1e5) { // 赢了~
                cout << "You win ^_^" << endl;
                break;
            }
            cout << "x score: " << ans << "     o score: " << -ans << endl; // 输
出分数
            cout << endl << "AI is thinking..." << endl;
            ma[0] = -1e9; mi[0] = 1e9; // 初始化兄弟节点
            ans = minimax(c, 0); // min_max搜索
            system("cls"); // 清屏
            a[x[0]][y[0]] = c; // AI下棋
            c = (c == 'x' ? 'o' : 'x'); // 下一步的棋子
        }
```

```
        else if (m == '2') { // 后手
            ma[0] = -1e9; mi[0] = 1e9; // 初始化兄弟节点
            int ans = minimax(c, 0); // min_max搜索
            system("cls"); // 清屏
            a[x[0]][y[0]] = c; // AI下棋
            c = (c == 'x' ? 'o' : 'x'); // 下一步的棋子
            print(); // 输出棋盘
            ans = judge('x')-judge('o'); // 计算当前局势
            if (ans >= 1e5) { // 输了~
                cout << "You lose >_<" << endl;
                break;
            }
            cout << "x score: " << ans << "     o score: " << -ans << endl; // 输出分数
            if (x[0] != 0 or y[0] != 0) cout << endl << "AI's choice:  " << y[0] << " " << len-x[0]+1 << endl;  // 输出AI上一步的决策
            cout << endl << "Your choice:  ";
            cin >> yourx >> youry;
            while (a[len-youry+1][yourx] != ' ' or len-youry+1 > len or len-youry+1 < 1 or youry > len or youry < 1) { // 判断输入是否合理
                cout << "You can't take this place! Choose another one:  ";
                cin >> yourx >> youry;
            }
            a[len-youry+1][yourx] = c; // 注意是二维坐标，不是矩阵坐标
            system("cls"); // 清屏
            print(); // 输出棋盘
            c = (c == 'x' ? 'o' : 'x'); // 下一步的棋子
            ans = judge('x')-judge('o'); // 计算当前局势
            if (-ans >= 1e5) { // 赢了~
                cout << "You win ^_^" << endl;
                break;
            }
            cout << "x score: " << ans << "     o score: " << -ans << endl; // 输出分数
            cout << endl << "AI is thinking..." << endl;
        }
    }
    if (t == 0) cout << "Battle end~" << endl; // 棋盘下满，和棋
```

创新点：

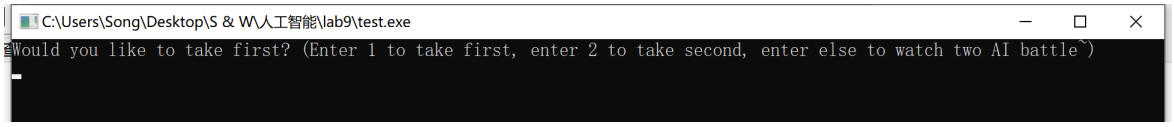
规划深搜：可以看出五子棋初始棋形下在棋盘中央地区，因此 AI 选择继续下在棋盘中央地区，和自己原有的棋形连接，或者限制玩家的棋形，评分都会比下在边缘地方要高。因此在 DFS 时先搜索棋盘中部地区，再搜索边缘地区，有利于 alpha-beta 剪枝算法对边缘地区基本不可能的下法进行剪枝

```
    for (int i=5; i<=len-4; i++) { // 先搜索棋盘中部
        for (int j=5; j<=len-4; j++) {
            if (a[i][j] == ' ' and vis[i][j] == 0) {
                vis[i][j] = 1;
                //...
    for (int i=1; i<=len; i++) { // 再搜索外部，下面代码完全相同~
        for (int j=1; j<=len; j++) {
            if (a[i][j] == ' ' and vis[i][j] == 0) {
                vis[i][j] = 1;
                //...
```

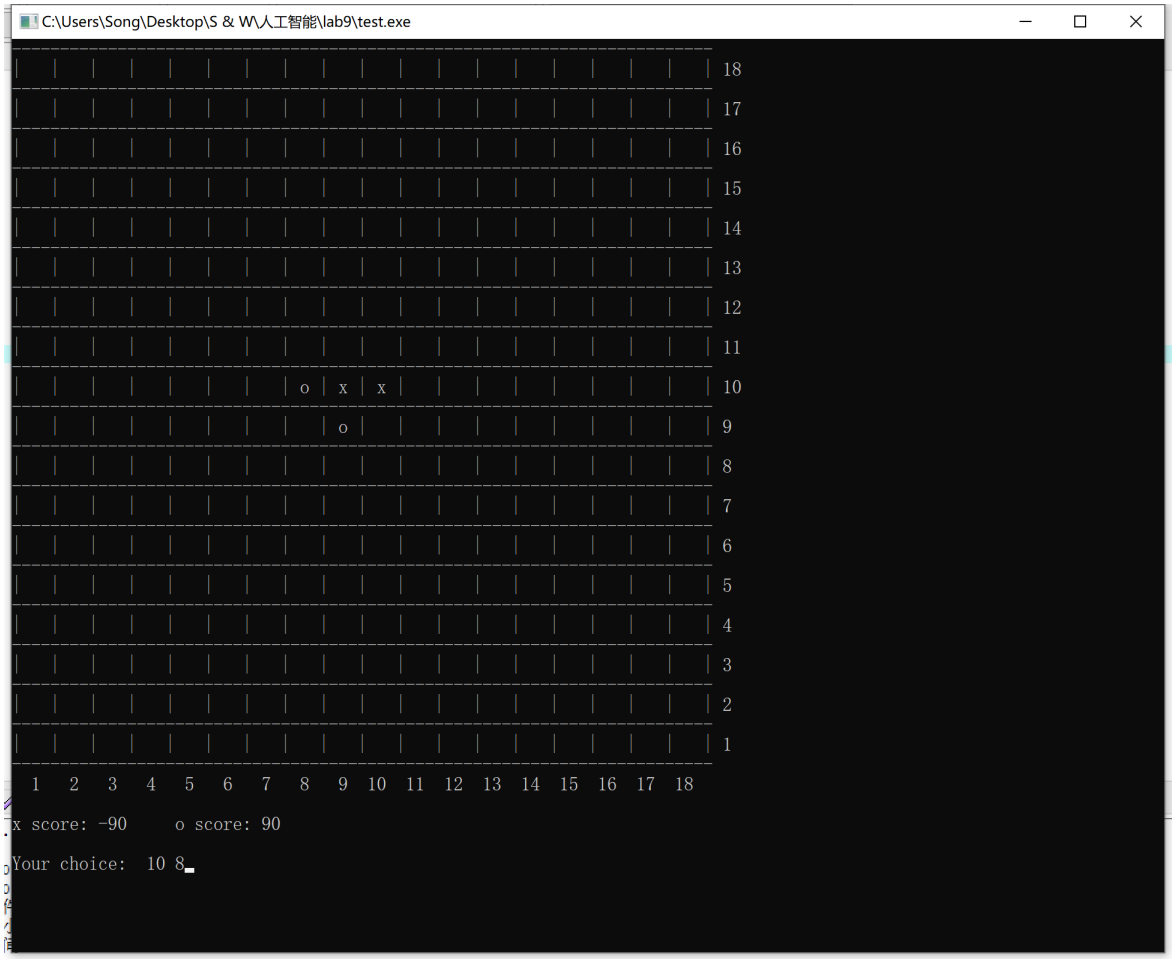
AI vs AI：既然开发了玩家 vs AI，那么开发个 AI vs AI，玩家作为旁观者（观察 AI 下棋是否合理）也是个不错的玩法（纯属娱乐）

实验结果以及分析：

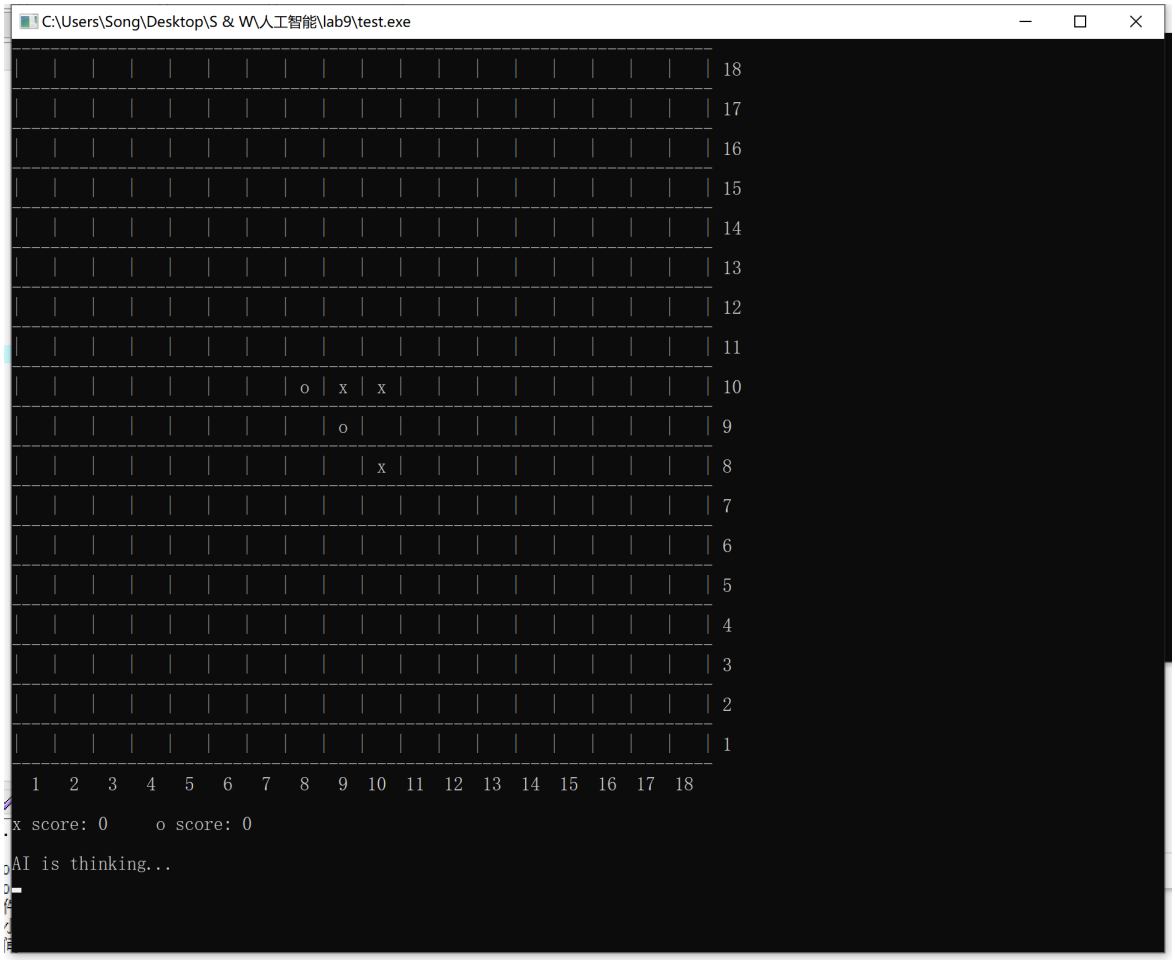
程序刚打开页面：输入 1 先手，输入 2 后手，其他输入旁观：



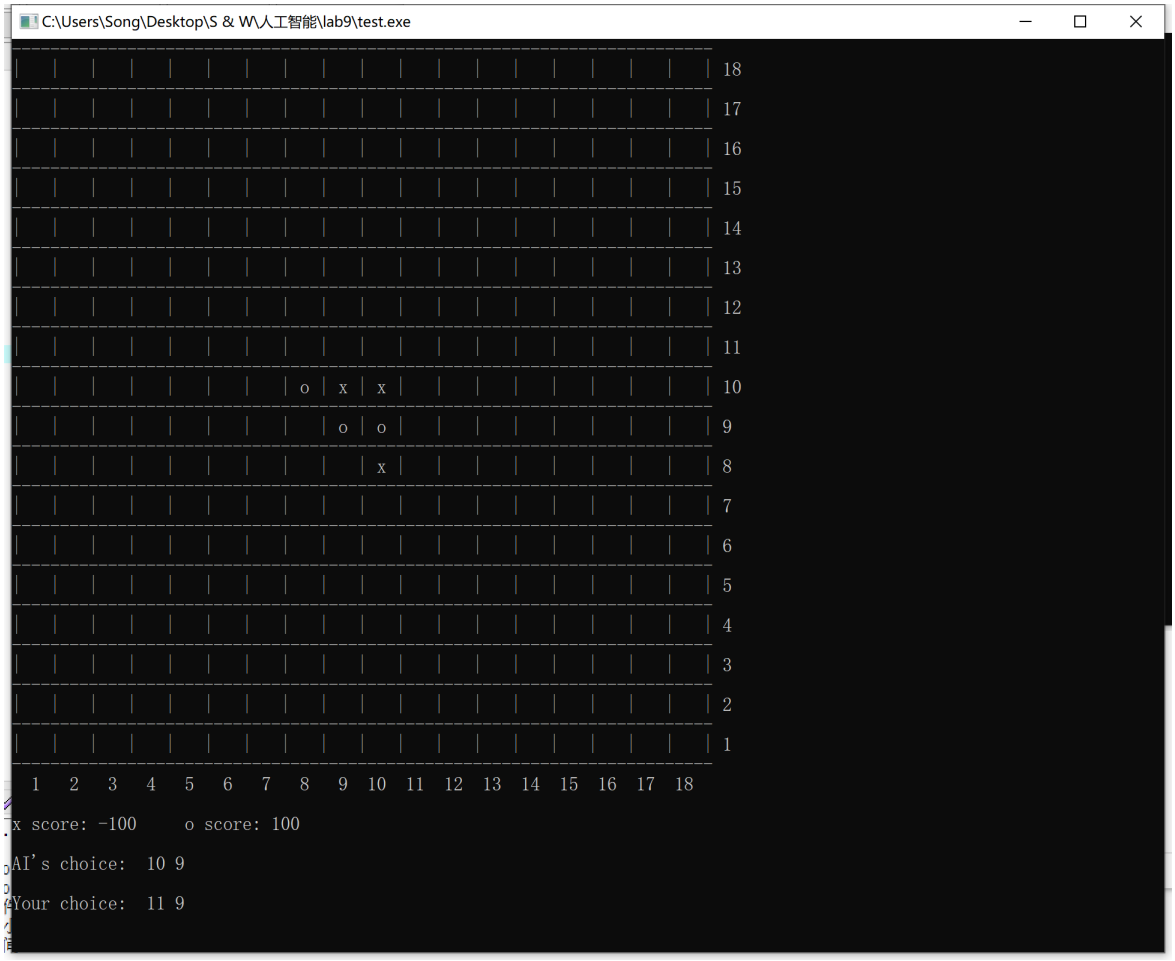
输入 1 进行先手测试，程序输出初始棋盘以及初始得分，提示玩家输入下棋位置**二维坐标**（即棋盘左下角为原点，输入 x,y 坐标）



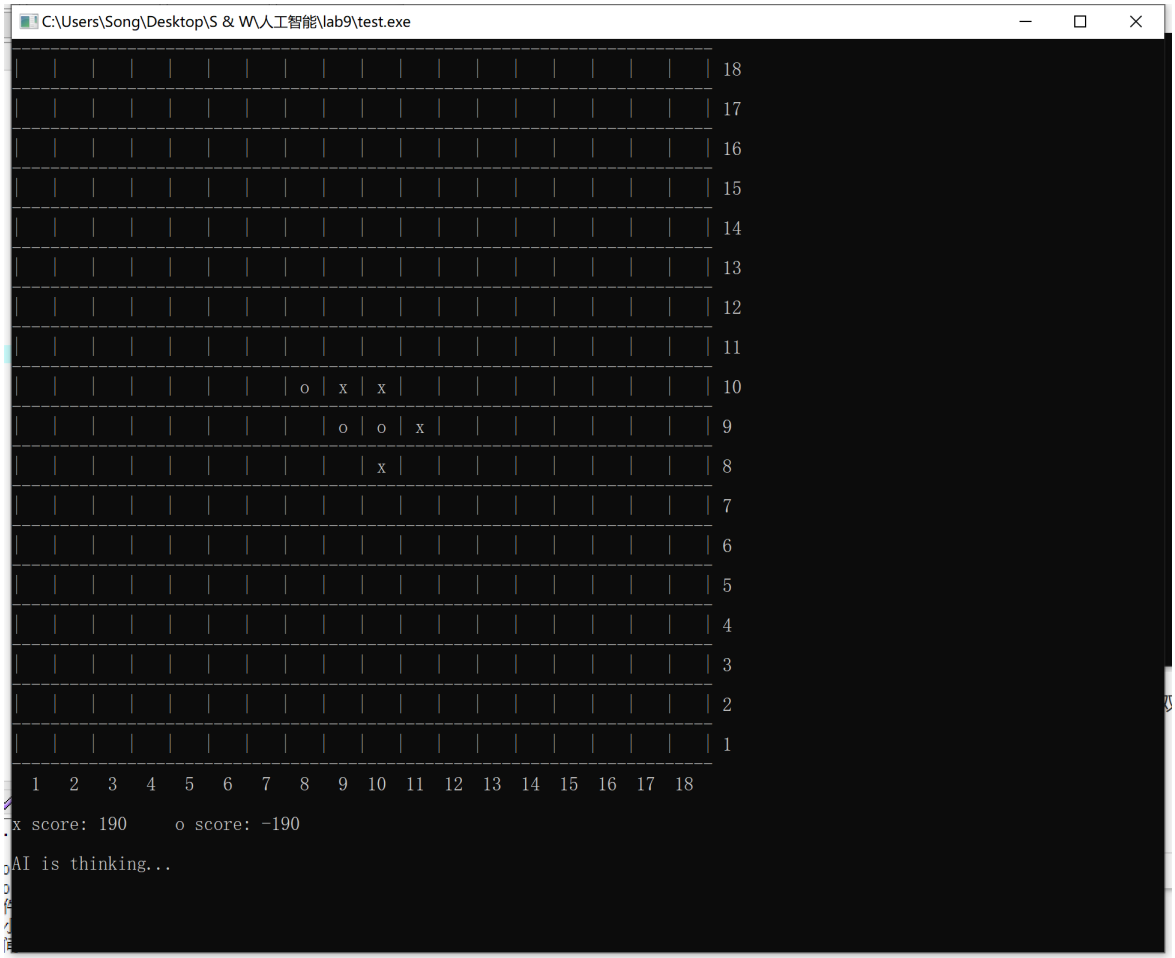
玩家输入 10 8 之后回车，可以看见（10，8）位置下了棋子 'x'，并输出新的评分（此时双方都是一组眠二，因此场面评分为 0）：



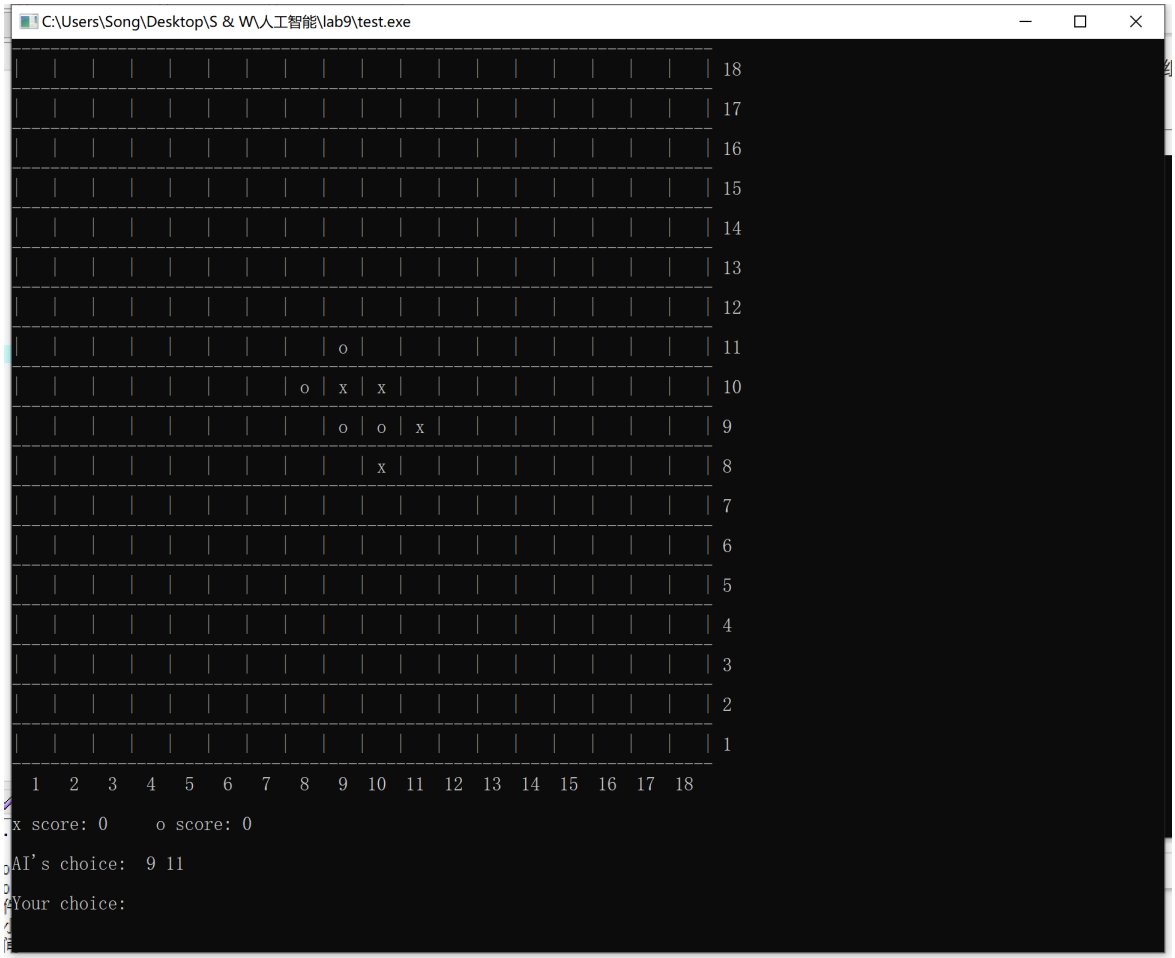
AI 下棋（10，9），输出新的评分（AI 一组活二、一组眠二，玩家一组眠二，因此 AI 领先 100 分）：



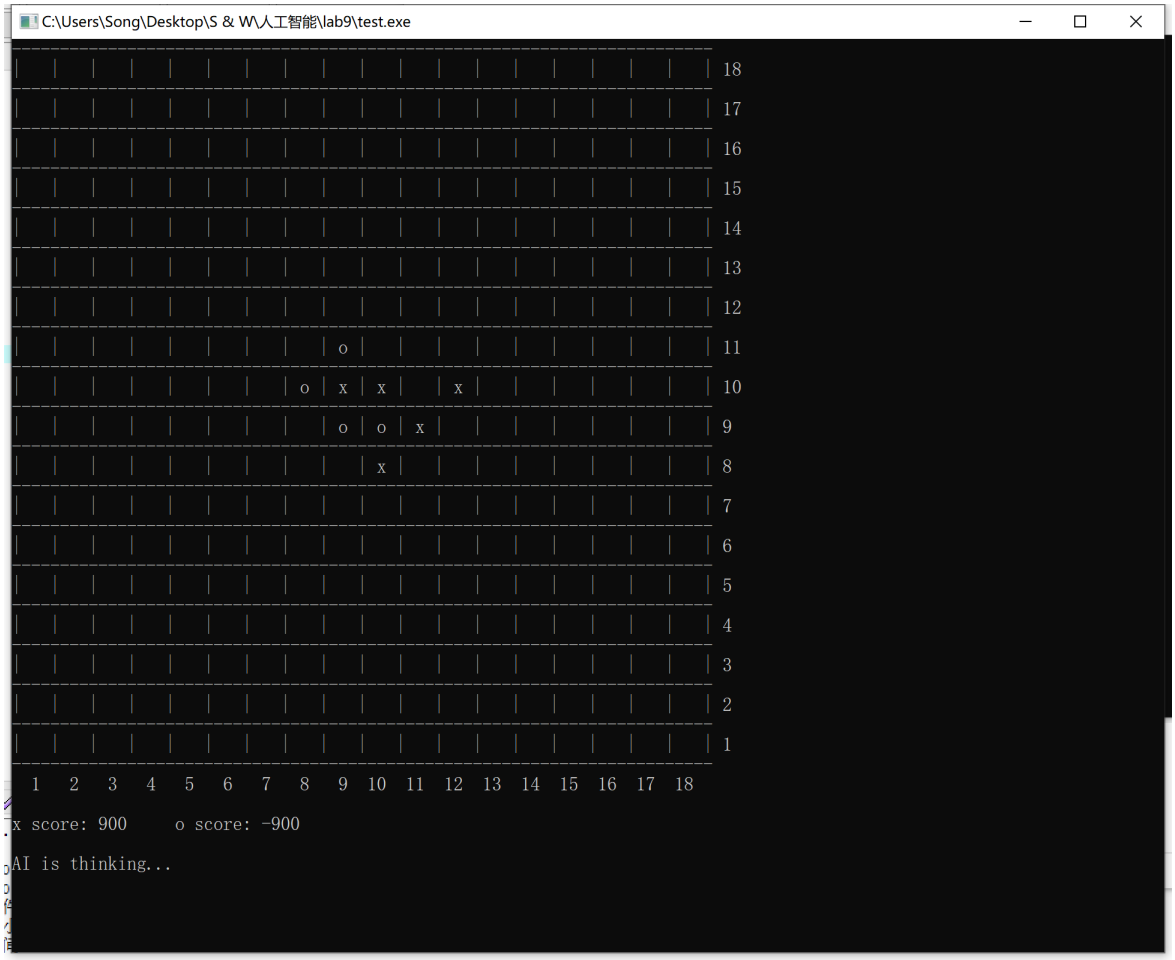
玩家输入 11 9 后回车，可以看见（11，9）位置下了棋子 'x'，并输出新的评分（AI 两组活二，玩家一组眠二，两组活二，因此玩家领先 190 分）：



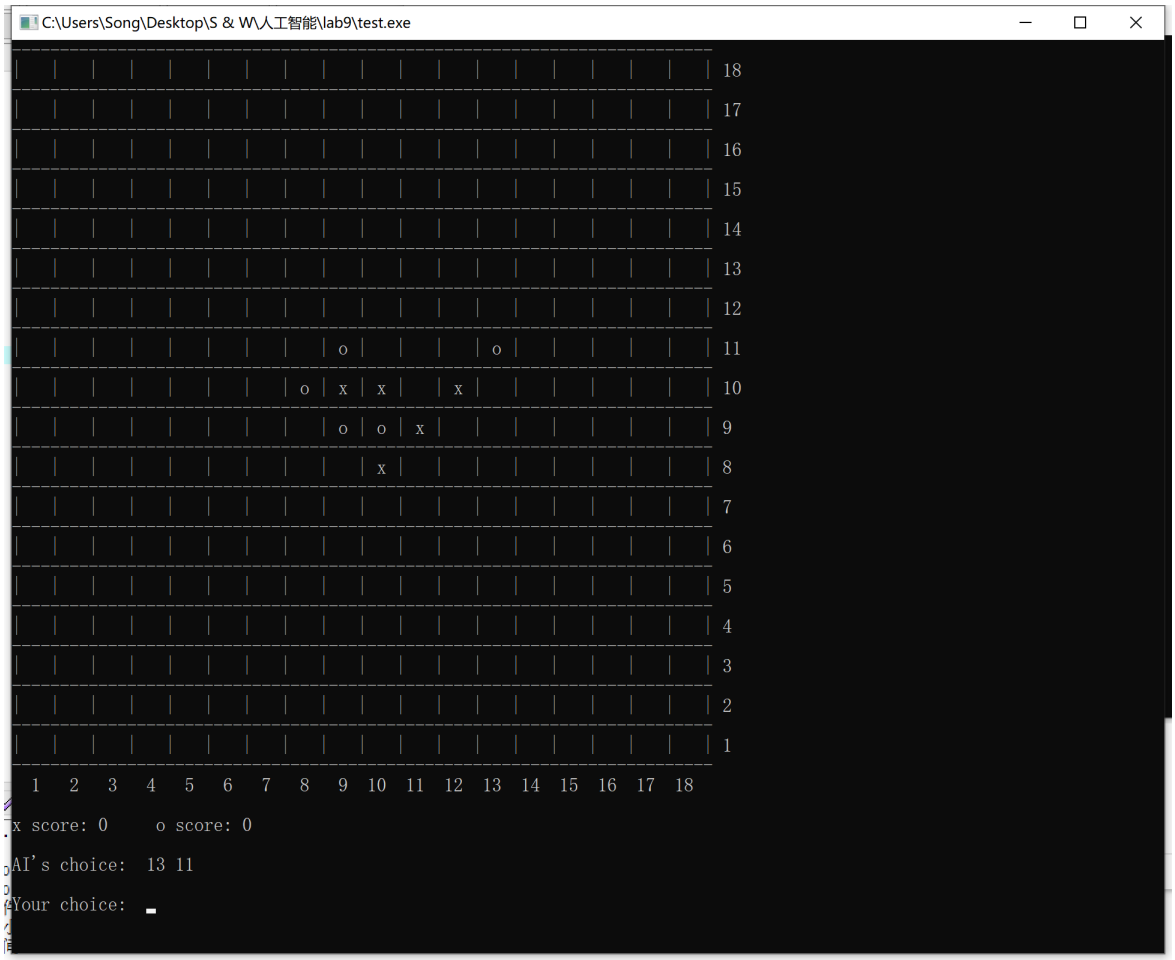
AI 下棋（9，11），输出新的评分（此时双方都是一组活二、一组眠二，因此场面评分为 0）：



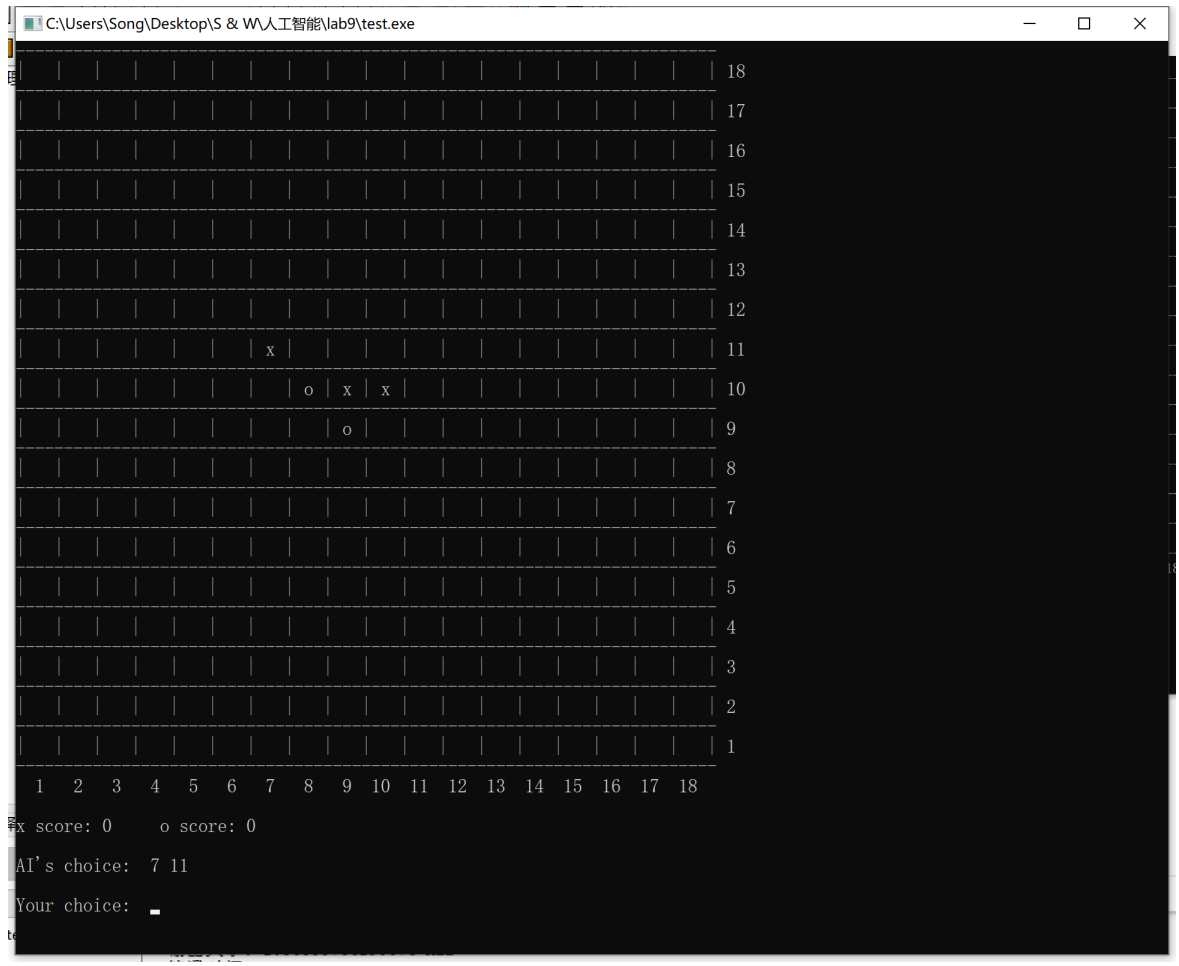
玩家输入 12 10 后回车，可以看见（12，10）位置下了棋子 'x'，并输出新的评分（AI 一组活二、一组眠二，玩家一组眠二，一组活三，因此玩家领先 900 分）：



AI 下棋（13，11），输出新的评分（AI 一组活二、一组眠二，玩家一组眠二，一组眠三，根据估值函数，场面评分为 0）：



程序刚开始时输入 2，开启后手模式，可以看出 AI 已在（7，11）处先下棋 'x'，输出此时评分



程序刚开始时输入其它任意输入，观看 AI vs AI，最终结果如下：