五子棋实验报告

学号：09016423

姓名：韩杰

设计思路

基本流程：

估值函数--->搜索算法--->胜负判断

核心算法

估值算法

思路：

不同的棋型，其优先级不同。例如，四个棋子连成一线且还能继续落子的棋型（活四）显然要比只有三个棋子连成一线（活三或死三）好。要使 AI 正确地做出这种判断，就要把第一种棋型的估值设高。以此类推，设计出估值函数。

其次，估值时要从四个方向考虑（水平、竖直、左斜、右斜）。

根据特定棋型（活四，死四诸如此类的）进行分值设置从而进行估值，也就是说，每走一步棋，都对当前棋子的四个方向进行搜索判断当前的棋型属于哪一种，进而得出分值，再调用搜索算法进行判断

棋型名称	棋型模式	估值
活四	?AAAA?	300000
死四 A	AAAA?	2500
死四 B	AAA?A	3000
死四 C	AA?AA	2600
活三	??AAA??	3000
死三 A	AAA??	500
死三 B	?A?AA?	800
死三 C	A??AA	600
死三 D	A?A?A	550
活二	???AA???	650
死二 A	AA???	150
死二 B	??A?A??	250
死二 C	?A??A?	200

图 1

搜索算法

思路：

加入了 alpha-beta 剪枝的极大极小搜索算法，虽然一定程度上减少了复杂度，但是效率其实还是不太高，因为一旦搜索层数加大，效率也会变得很低，最重要的是这样的 AI 智力不太高，所以我想的是可以把搜索范围减少到以当前棋子为中心的一个 9x9 的方块（如下图 2 所示），其次，可以使用置换表，以避免重复搜索，提高效率。此外，我还有一个想法，就是说，可以设定一个 AI 思考的时间范围来适当地增大搜索层数（比如说，设定 AI 思考时间为 0.5s，那么只要搜索时间不超过 0.5s，就继续搜索）。

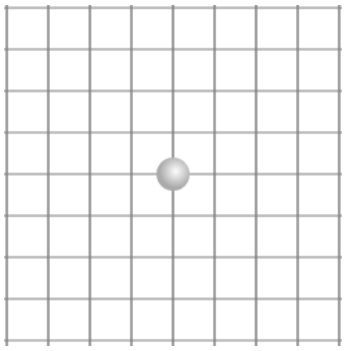

图 2

Alpha-beta 搜索算法

```
int AlphaBeta(int depth, int alpha, int beta) {
    if (depth == 0)
        如果深度为 0，返回当前局势估值

    while 还能继续走
        走一步棋
        从对手角调用 -AlphaBeta(depth - 1, -beta, -alpha);递归搜索，记为 val
        撤销走的棋
        if (val >= beta) {
            return beta;
        }
        if (val > alpha) {
            alpha = val;
        }
    }
    return alpha;
}
```

胜负判断算法

思路：

在棋局的胜负是根据最后一个落子的情况来判断的。此时需要查看四个方向，即以该棋子为出发点的水平，竖直和两条分别为 45 度角和 135 度角的线，看在这四个方向上的其它棋子是否能和最后落子构成连续五个棋子，如果能的话，则表示这盘棋局已经分出胜负。我现在想的是把所有的赢法储存在一个一维数组中，每下一步棋都遍历所有赢法数组（距以落点为中心的边长为 8 的正方形，如图 2 所示）