Report I

09016428 SongZixing

1 Introduction

Gomoku, also called Gobang or Five in a Row, is an abstract strategy board game. It is traditionally played on a Go board, using 15×15 of the 19×19 grid intersections. To make it easier, the Gomoku we dissucss in this report is based on Free-style Gomoku which means a row of five or more stones is considered as a win.

2 Algorithm

After some initial research, I find that there are various kinds of algorithms leading to the implement of Gomoku AI. And in Report 1, I wil just cover the most prevalent and fundmental ones and in the following reports and introduce how I transform them to sources codes. I am going to optimize the current algorithms in the terms of both time and space and even replace them with more advanced and efficient ones.

2.1 Minimax

2.1.1 Overview

Minimax is a decision rule widely used in game theory for minimizing the possible loss for a worst case scenario, which ,technically speaking, means the maximum loss. Likewise, it is referred to as "maximum" , to maximize the minimum gain when dealing with gains.

In the field of zero-sum games the maximin rule or strategy states the following:

> The best possible play for player 1 in a given state is the the one that minimizes the best possible play for player 2 from that state;

This means that we can conceive an algorithm that calculates the best possible play from a given state by looking for the worst best possible play from all of the possible subsequent moves. Of course this does not apply for terminal moves, that is, moves where the game ends, where their value is simply the outcome of the game in that particular state.

2.1.2 Minimax algorithm with alternate moves

A minimax algorithm is a recursive algorithm for choosing the next move in an n-player game, usually a two-player game. A value is associated with each position or state of the game. This value is computed by means of a position evaluation function and it indicates how good it would be for a player to reach that position. The player then makes the move that maximizes the minimum value of the position resulting from the opponent's possible following moves. If it is A's turn to move, A gives a value to each of their legal moves.

2.1.3 Pseudocode

```
01 function minimax(node, depth, maximizingPlayer)
02     if depth = 0 or node is a terminal node
03         return the heuristic value of node

04     if maximizingPlayer
05         bestValue := −∞
06         for each child of node
07             v := minimax(child, depth − 1, FALSE)
08             bestValue := max(bestValue, v)
09         return bestValue

10     else    (* minimizing player *)
11         bestValue := +∞
12         for each child of node
13             v := minimax(child, depth − 1, TRUE)
14             bestValue := min(bestValue, v)
15         return bestValue
```

```
// Initial call for maximizing player
minimax(origin, depth, TRUE)
```

2.1.4 Analysis

The minimax function returns a heuristic value for leaf nodes (terminal nodes and nodes at the maximum search depth). Non leaf nodes inherit their value, *bestValue*, from a descendant leaf node. The heuristic value is a score measuring the favorability of the node for the maximizing player. Hence nodes resulting in a favorable outcome, such as a win, for the maximizing player have higher scores than nodes more favorable for the minimizing player. The heuristic value for terminal (game ending) leaf nodes are scores corresponding to win, loss, or draw, for the maximizing player. For non terminal leaf nodes at the maximum search depth, an evaluation function estimates a heuristic value for the node. The quality of this estimate and the search depth determine the quality and accuracy of the final minimax result.

However, the time the Minimax algorithm consume is supposed to increase exponentially as the size of input expands, especially as the game goes on further. As a consequence, Minimax algorithm is theoreical but by no means practical. But we can do some optimization based on this algorithm, which reduces the time it takes to a great extent.

2.2 Negamax

2.2.1 Overview

Negamax search is a variant form of Minimax search that relies on the zero-sum property of a two-player game.

This algorithm relies on following fact.

$$max(a, b) = -min(-a, -b)$$

This obvious equation can simplify the implementation of the Minimax algorithm. More precisely, the value of a position to player A in such a game is the negation of the value to player B. Thus, the player on move looks for a move that maximizes the negation of the value resulting from the move: this successor position must by definition have been valued by the opponent. The reasoning of the previous sentence works regardless of whether A or B is on move. This means that a single procedure can be used to value both positions. This is a coding simplification over minimax, which requires that A selects the move with the maximum-valued successor while B selects the move with the minimum-valued successor.

Actually, the Negamax is a variation or perhaps an extension of the Minimax rules. It relies on the fact that the maximum of two objects is the negation of the minimum of the negatives of the objects. It states the following:

> If from a given state, the minimum value of the best plays of the possible adjacent moves is P, then the value of the state is -P.

2.2.2 Pseudocode

```
01 function negamax(node, depth, color)
02     if depth = 0 or node is a terminal node
03         return color * the heuristic value of node

04     bestValue := -∞
05     foreach child of node
06         v := -negamax(child, depth - 1, -color)
07         bestValue := max( bestValue, v )
08     return bestValue
```

```
//Initial call for Player A's root node
rootNegamaxValue := negamax( rootNode, depth, 1)
rootMinimaxValue := rootNegamaxValue
```

```
//Initial call for Player B's root node
rootNegamaxValue := negamax( rootNode, depth, -1)
rootMinimaxValue := -rootNegamaxValue
```

2.2.3 Analysis

The root node inherits its score from one of its immediate child nodes. The child node that ultimately sets the root node's best score also represents the best move to play. Although the negamax function shown only returns the node's best score as *bestValue*, practical negamax implementations may also retain and return both best move and best score for the root node.

Assuming basic negamax, only the node's best score is essential with non-root nodes. And the node's best move isn't necessary to retain nor return for those nodes.

2.3 Alpha-beta pruning

2.3.1 Introduction

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It is an adversarial search algorithm used commonly for machine playing of two-player games. It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

2.3.2 Core ideas

The algorithm maintains two values, alpha and beta, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially alpha is negative infinity and beta is positive infinity, and both players start with their worst possible score. Whenever the maximum score that the minimizing player(beta) is assured of becomes less than the minimum score that the maximizing player(alpha) is assured of (beta <= alpha), the maximizing player need not consider the descendants of this node as they will never be reached in actual play.

2.3.3 Minimax with alpha-beta pruning (Pseudocode)

```
01 function alphabeta(node, depth, α, β, maximizingPlayer)
02      if depth = 0 or node is a terminal node
03          return the heuristic value of node
04      if maximizingPlayer
05          v := -∞
06          for each child of node
07              v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
08              α := max(α, v)
09              if β ≤ α
10                  break (* β cut-off *)
11          return v
12      else
13          v := +∞
14          for each child of node
15              v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
16              β := min(β, v)
17              if β ≤ α
18                  break (* α cut-off *)
19          return v

//Initial call
```

```
alphabeta(origin, depth, -∞, +∞, TRUE)
```

2.3.4 Negmax with alpha-beta pruning(Pseudocode)

```
01 function negamax(node, depth, α, β, color)
02     if depth = 0 or node is a terminal node
03         return color * the heuristic value of node

04     childNodes := GenerateMoves(node)
05     childNodes := OrderMoves(childNodes)
06     bestValue := -∞
07     foreach child in childNodes
08         v := -negamax(child, depth - 1, -β, -α, -color)
09         bestValue := max( bestValue, v )
10         α := max( α, v )
11         if α ≥ β
12             break
13     return bestValue


//Initial call for Player A's root node
rootNegamaxValue := negamax( rootNode, depth, -∞, +∞, 1)
```

2.3.5 Asymptotic analysis

With an (average or constant) branching factor of b, and a search depth of d plies, we can reach the following conclusion that $O(b^d)$ is the worst-case performance while $O(\sqrt{b^d})$ is the best case performance.

We choose to omit the proof due to the complexity of this problem.

3 Implement

In this part, I will mainly talk about how I implement the Gomoku AI step by step, using C++.

3.1 Preparation

First, I define a struct called MOVE as follow to represent all the moves the game will involve.

```
typedef struct MOVE {
    int x; int y;//coordinates
    MOVE() {// initial coordinates
        x = -1;
        y = -1;
    }
    MOVE(int tx, int ty) {// set move function
        x = tx;
        y = ty;
    }
};
```

Then I define a Class called Gomoku as follow to represent the current state of the 15*15 grid.

```
class Gomoku
{
public:
    Gomoku();
    ~Gomoku();

    bool VaildMove(int x, int y);
    void SetMove(int x,int y,int color);
    void applyMove(MOVE newMove);


    int WinOrLose();
    bool WhetherWin(int x,int y,int color);
    bool canContinue();
    bool isGameOver();

    void Print();

    void StartGame(int color);
    void StartAIGame(int color, int depth);

    vector<MOVE> GenerateLegalMoves();

    //determine the all the possible structures
    int WinFive(int x, int y);
    int  AliveFour(int x, int y);
    int  DeadFourA(int x, int y);
    int  DeadFourB(int x, int y);
    int  DeadFourC(int x, int y);
    int AliveThree(int x, int y);
    int DeadThressA(int x, int y);
    int DeadThressB(int x, int y);
    int DeadThressC(int x, int y);
    int DeadThressD(int x, int y);
```

```cpp
    int AliveTwo(int x, int y);
    int DeadTwoA(int x, int y);
    int DeadTwoB(int x, int y);
    int DeadTwoC(int x, int y);

    //evaluate function
    double EvaluatePoint(int i, int j);
    double EvaluateState();

    //AlphaBeta_Minimax function
    MoveValue minMax(double alpha, double beta, int maxDepth, int player);


    private:
    int Grid[15][15];// the grid
    int CurrentColor;//current player
    int AIColor;//AI player
    double mark;//current evaluate value

};
```

Then here is the `EvaluateState();`

```cpp
double Gomoku::EvaluatePoint(int i, int j)
{
    if(Grid[i][j]==CurrentColor)

    return  WinFive(i,j)* (1000000000000 + 300000000 * 3) +AliveFour(i, j) *
(300000000 + 30000 * 3) + DeadFourA(i, j) * 25000 + DeadFourB(i, j) * 30000 +
DeadFourC(i, j) * 26000 + AliveThree(i, j) * 20000 + DeadThressA(i, j) * 500 +
DeadThressB(i, j) * 800 + DeadThressC(i, j) * 600 + DeadThressD(i, j) * 550 +
AliveTwo(i, j) * 650 + DeadTwoA(i, j) * 150 + DeadTwoB(i, j) * 250 + DeadTwoC(i,
j) * 200;

    else if (Grid[i][j] = CurrentColor % 2 + 1)
    {
        double res = 0; CurrentColor = CurrentColor % 2 + 1;
        res= WinFive(i, j) * (1000000000000) +AliveFour(i, j) * (300000000 )+
DeadFourA(i, j) * 25000 + DeadFourB(i, j) * 30000 + DeadFourC(i, j) * 26000 +
AliveThree(i, j) * 100000 + DeadThressA(i, j) * 500 + DeadThressB(i, j) * 800 +
DeadThressC(i, j) * 600 + DeadThressD(i, j) * 550 + AliveTwo(i, j) * 650 +
DeadTwoA(i, j) * 150 + DeadTwoB(i, j) * 250 + DeadTwoC(i, j) * 200;
        CurrentColor = CurrentColor % 2 + 1;
        return res;
    }
}
```

```cpp
double Gomoku::EvaluateState()
{
    double  res = 0; int i, j;
    for(i=0;i<15;i++)
        for(j=0;j<15;j++)
        {
            if (Grid[i][j] == (AIColor)) res += EvaluatePoint(i, j);
            else if (Grid[i][j] == (AIColor % 2 + 1)) res -= EvaluatePoint(i, j);
        }
    return res;
}
```

To make it more readable, I omit all the other functions related to the preparation for the implement of the key algorithm `AlphaBeta_Minimax function()`.

Before we reach the key algorithm, I have to admit the fact that the pseudocode related to the Minimax with alpha-beta pruning do not return the decided move. So I define another struct to store both the value and the move. Here is the struct called `MoveValue`.

```cpp
struct MoveValue {

    double returnValue;//value
     MOVE returnMove;//move

     MoveValue() {
        returnValue = 0;
    }

     MoveValue(double returnValuepara) {
        returnValue = returnValuepara;
    }

     MoveValue(double returnValuepara, MOVE returnMovepara) {
        returnValue = returnValuepara;
        returnMove = returnMovepara;
    }

     bool operator ==(MoveValue& t) {
        if (t.returnValue == returnValue && t.returnMove.x ==
returnMove.x&&t.returnMove.y == returnMove.y) return true;
        else return false;
    }

};
```

3.2 Key algorithm implement

Here is the final `AlphaBeta_Minimax function()` .

```cpp
MoveValue Gomoku::minMax(double alpha, double beta, int maxDepth, int player)
{
    vector<MOVE> moves = GenerateLegalMoves();
    vector<MOVE>::iterator movesIterator = moves.begin();

    bool isMaximizer = player == AIColor ? true : false;

    if (maxDepth == 0 || isGameOver()) {
        MoveValue res(EvaluateState());
        return res;
    }
    MoveValue returnMove;

    if (isMaximizer) {

    double a = DBL_MIN; MOVE newMOVE(-1, -1);
    MoveValue DefaultMove(a, newMOVE);
    MoveValue bestMove(DefaultMove);


        while (movesIterator != moves.end()) {

            MOVE currentMove = *movesIterator;
            movesIterator++;
            applyMove(currentMove);
            returnMove = minMax(alpha, beta, maxDepth - 1, player % 2 + 1);
            {
                Grid[currentMove.x][currentMove.y] = 0;
                CurrentColor = CurrentColor % 2 + 1;
            }
            //board.undoLastMove();
            if ((bestMove.returnValue == DefaultMove.returnValue) ||
(bestMove.returnValue < returnMove.returnValue)) {
                bestMove.returnValue = returnMove.returnValue;
                bestMove.returnMove = currentMove;
            }
            if (bestMove.returnValue > alpha) {
                alpha = bestMove.returnValue;
            }
            if (beta <= alpha) {
                break; // pruning
            }
        }
        return bestMove;
```

```
        }
        else {

            double a = DBL_MAX; MOVE newMOVE(-1, -1);
            MoveValue DefaultMove(a, newMOVE);
            MoveValue bestMove(DefaultMove);

            while (movesIterator != moves.end()) {
                MOVE currentMove = *movesIterator;
                movesIterator++;

                applyMove(currentMove);
                returnMove = minMax(alpha, beta, maxDepth - 1, player% 2 + 1);
                {
                    Grid[currentMove.x][currentMove.y] = 0;
                    CurrentColor = CurrentColor % 2 + 1;
                }
                //board.undoLastMove();
                if ((bestMove == DefaultMove) || (bestMove.returnValue >
returnMove.returnValue)) {
                    bestMove.returnValue = returnMove.returnValue;
                    bestMove.returnMove = currentMove;
                }
                if (bestMove.returnValue  < beta) {
                    beta = bestMove.returnValue;
                }
                if (beta <= alpha) {
                    break; // pruning
                }
            }
            return bestMove;
        }
    }
```

Results

Due to limited time, I fail to work out a GUI to illustrate the intelligence level of my Gomoku. Instead I simply make it compete with myself first.

In the test, AI plays the black while I play the white.

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o * o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o * + o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
6 6请按任意键继续. . .
AI takes the coordinate: 6, 6
1900
0请按任意键继续. . .
```

```
C:\Users\Administrator\Desktop\Gomoku_AI\Debug\Gomoku_AI.exe

o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o * o o o o o o o o
o o o o o o o * + o o o o o o
o o o o o o o + o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
6 5请按任意键继续. . .
```



```
C:\Users\Administrator\Desktop\Gomoku_AI\Debug\Gomoku_AI.exe

o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o * o o o o o o o
o o o o o o o * + o o o o o o
o o o o o o o + o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
6 5请按任意键继续. . .
AI takes the coordinate: 6, 5
300
0请按任意键继续. . .
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o * * o + o o o o o
o o o o o o o * + o o o o o
o o o o o o o + o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
5 8请按任意键继续. . .
AI takes the coordinate: 5, 8
-1200
-1200请按任意键继续. . .
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o * o o o o o o
o o o o o o * * o + o o o o o
o o o o o o o * * + o o o o o
o o o o o o o o + + o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o * o * o o o o o o
o o o o o * * + + o o o o o o
o o o o o * * + o o o o o o o
o o o o o o o + + o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o + o o o o o o o o
o o o o o o * o * o o o o o o
o o o o o * * + + o o o o o o
o o o o o * * + o o o o o o o
o o o o o o o + + o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
3 6请按任意键继续. . .
AI takes the coordinate: 3, 6
1200
-200请按任意键继续. . .
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o * * o o o o o o o o
o o o o o o + o o o o o o o o
o o o o o o * + * o o o o o o
o o o o o * * + + o o o o o o
o o o o o o * * + o o o o o o
o o o o o o o o + + o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o * * o o o o o o o o
o o o o o o + o o o o o o o o
o o o o o o * + * o o o o o o
o o o o o o * * + + o o o o o o
o o o o o o * * + + o o o o o
o o o o o o o o + + * o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```

```
C:\Users\Administrator\Desktop\Gomoku_AI\Debug\Gomoku_AI.exe

o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o * * o o o o o o o o
o o o o o o + o o o o o o o o
o o o o o o * + * o o o o o o
o o o o o * * + + o o o o o o
o o o o o o * * + + o o o o o
o o o o o o o + + * o o o o o
o o o o o o o o + o o o o o o
o o o o o o o * o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```

```
C:\Users\Administrator\Desktop\Gomoku_AI\Debug\Gomoku_AI.exe

o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o * * o o o o o o o o
o o o o o o + o o o o o o o o
o o o o o o * + * o o o o o o
o o o o o * * + + o o o o o o
o o o o o o * * + + + o o o o
o o o o o o o + + * o o o o o
o o o o o o o o + o o o o o o
o o o o o o o * * o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o * * o o o o o o o o
o o o o o o + o o o o o o o o
o o o o o o * + * o o * o o o
o o o o o * * + + o + o o o o
o o o o o o * * + + + o o o o
o o o o o o o o + + * o o o o
o o o o o o o o + o o o o o o
o o o o o o * * o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o * * o o o o o o o o
o o o o o o + o o o o o o o o
o o o o o o * + * o o * o o o
o o o o o * * + + o + + o o o
o o o o o o * * + + + o o o o
o o o o o o * o + + * o o o o
o o o o o o o o + o o o o o o
o o o o o o o * * o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o * * o o o o o o o o
o o o o o o + o o o o o o o o
o o o o o o * + * o o * * o o
o o o o o * * + + o + + o o o
o o o o o o * * + + + o o o o
o o o o o o * o + + * o o o o
o o o o o o + o + o o o o o o
o o o o o o o * * o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```

```
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o * * o o o o o o o o
o o o o o o + o o o o o o o o
o o o o o o * + * o o * * o o
o o o o o * * + + o + + o o o
o o o o o o * * + + + o o o o
o o o o o o * o + + * o o o o
o o o o o o + + + * o o o o o
o o o o o o o * * o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
o o o o o o o o o o o o o o o
please input the coordinate of white:
```
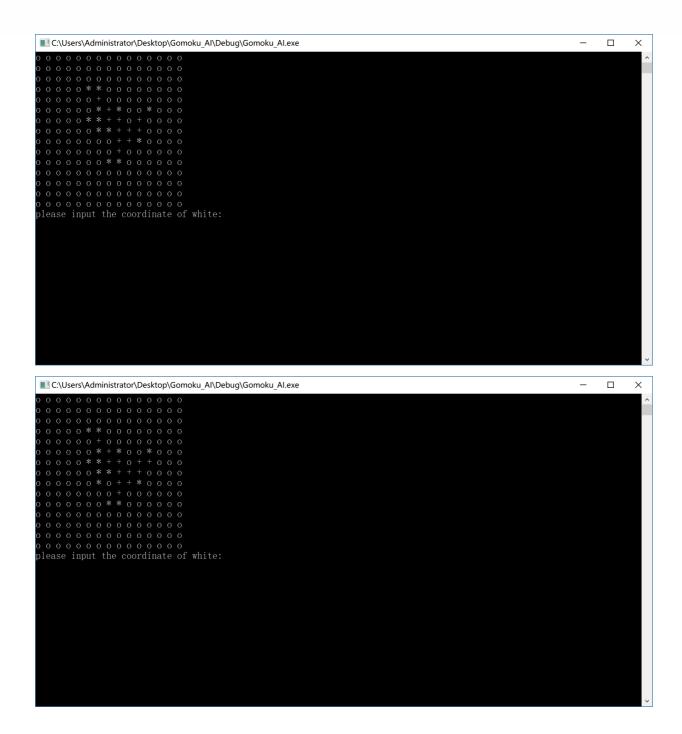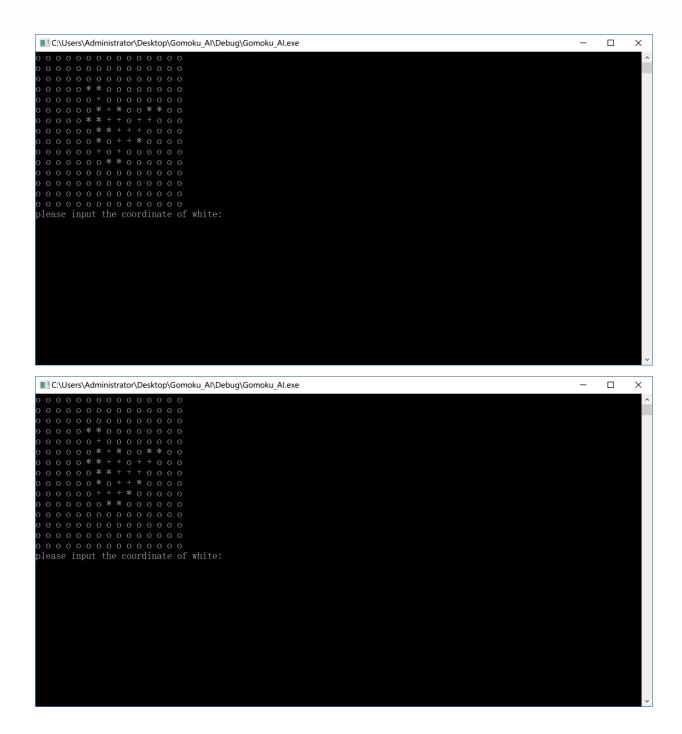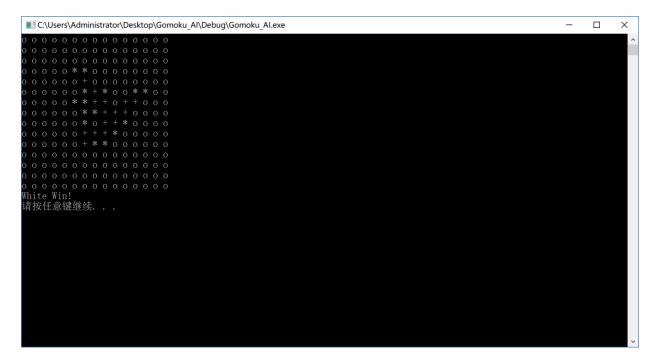
Though I win at last, I have to admit that the AI indeed has some sort of intelligence to some degree. I will say that the `evaluate()` function plays an important role in the IQ level of AI. Also, the search depth matters as well.

Improvement

After some tests, I find that the AI will take an increasing amount of time as the game goes on due to the growing size of search tree. So I will use some optimization algorithms to improve my AI performance in the terms of time it consumes, using Aspiration search or Transposition table.