Gomoku

Ver 2.1

09016401
顾婷瑄
Botzone id: Gutingxuan

1. Old versions review

Gomoku2.0 is simply based on a MaxMin algorithm and although it can foresee future moves, it spends a lot of time on doing calculations so the deepest searching level it can reach is a mere three.

2. Literatures I read

[1] 王志水. 基于搜索算法的人工智能在五子棋博弈中的应用研究[D].山东：中国石油大学, 2006

[2] Blog: 五子棋基本棋型@我是老邱

五子棋估值算法@maxuewei2

3. Current idea of design

i. Essence

A function based on the MaxMin algorithm and the alpha-beta pruning.

ii. Origin of the idea

The paper I’ ve read.

iii. Expectation

The function can make tentative moves, evaluate the value of each state on both sides and then make the best choice. Furthermore, the time it spends to do the calculations will be decreased.

iv. Implementation

A MaxMin algorithm, alpha-beta pruning, new evaluation method.

v. Advantages of the design

With the alpha-beta pruning, the time spent on calculating can be decreased. I sort the values of the available positions before searching and only visit the first 12 positions, so the same number of searching nodes can lead to a deeper searching level. Since there is a time limit on botzone, so my current searching level is 6. And with the new evaluation method, the current board value will be calculated more accurately.

vi. Weakness

The alpha-beta pruning relies on the order of the positions it visits. If it goes to the worst position first, then the beta-pruning won’ t happen. So,

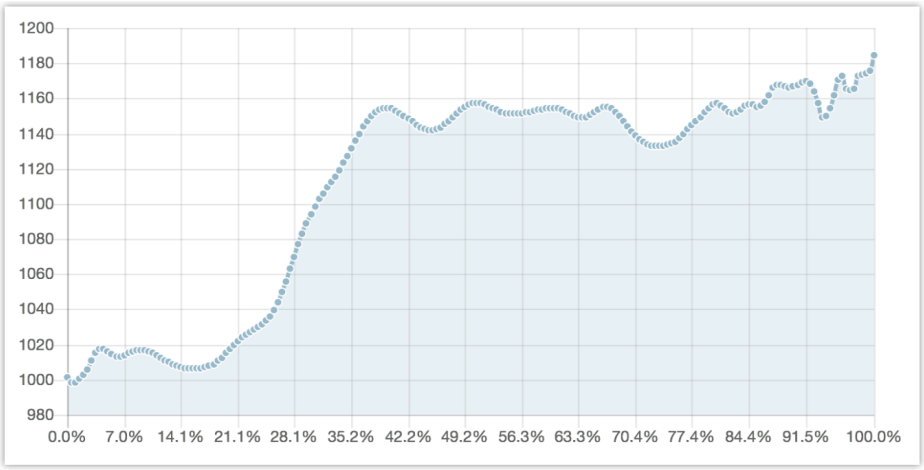
I have done the sorting before making attempted moves, which is to say the function will visit those most likely moves first. And to save some searching time, I choose to only visit the first twelve positions. So if there’ s any flaw in my evaluation method of each position, it will affect the alpha-beta pruning.

vii. Improvement directions

I will try to optimize the evaluation method and try to improve my algorithm.

4. Evaluation of the performance

Current ranking on Botzone: 20/37 (2018/5/3 20:00)



总趋势

The score increased a lot after I limit the number of the searching nodes. Because there is a time limit on botzone so if the time my gomoku program takes exceeds the time limit, it just loses the game.

5. Code structure and detailed implementations

i. Alpha-beta pruning

Explanation from wikipedia:

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 (Tic-tac-toe, Chess, Go, etc.). 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.

Pseudocode:

```
int AlphaBeta(int depth, int alpha, int beta)
{
    if (depth == 0)
    {
        return Evaluate();
    }
    GenerateLegalMoves();
    while (MovesLeft())
    {
        MakeNextMove();
        val = -AlphaBeta(depth - 1, -beta, -alpha);
        UnmakeMove();
        if (val >= beta)
        {
            return beta;
        }
        if (val > alpha)
        {
            alpha = val;
        }
    }
    return alpha;
}
```

My alpha-beta pruning part:

```
int count = 0;
for (int i = 0; i < k; i++)
{
    board[move1[list[i]][0]][move1[list[i]][1]] = choice;
    val = -AlphaBeta(deep - 1, -beta, -alpha);
    board[move1[list[i]][0]][move1[list[i]][1]] = -1;
    if (val >= beta)
    {
        return beta;
    }
    if (val > alpha)
    {
        alpha = val;
        if(deep == 8)
        {
            lastx = move1[list[i]][0];
            lasty = move1[list[i]][1];
        }
    }
    if(count++ >= 12)   break;
}
return alpha;
```

ii. New Evaluation method

In the former version, I choose the most valuable position score as current board value. Actually it's not very wise. In this version, I choose to collect all those positions that can detect a three-in-a-row or a four-in-a-row and add up all those scores as the total board value.

```
int valueAI=0,valueHM=0;
int move[225][2];
int k=0;
for(int x=0;x<15;x++)
    for(int y=0;y<15;y++)
    {
        if(board[x][y]==-1)
        {
            for(int i=1;i<=8;i++)
            {
                if(getLine(x,y,i,1)==0||getLine(x,y,i,1)==1)
                {
                    move[k][0]=x;
                    move[k][1]=y;
                    k++;
                    break;
                }
            }
        }
    }
for(int i=0;i<k;i++)
{
    valueAI+=Evaluate(move[k][0],move[k][0],choice);
    valueHM+=Evaluate(move[k][0],move[k][0],1-choice);
}
return valueAI - valueHM;
```

iii. Extra tricks

①

```
int a[k];
for(int i=0;i<k;i++)
{
    if(EvaluateA(move1[i][0],move1[i][1],choice)>EvaluateD(move1[i][0],move1[i][1],1-choice))
        a[i]=score2[move1[i][0]][move1[i][1]];
    else a[i]=score1[move1[i][0]][move1[i][1]];
}
int list[k];
for(int i=0;i<k;i++)
{
    int max=-1;
    int temp=0;
    for(int j=0;j<k;j++)
    {
        if(a[j]>max)
        {
            max=a[j];
            temp=j;
        }
    }
    list[i]=temp;
    a[temp]=-100;
}
```

Obviously, the position with a higher value is more likely to be chosen. So these lines are to sort the values of the available positions before the actual searching to enhance the ability of alpha-beta pruning.

②

```
if(count++ >= 12)    break;
```

This is to choose the first 12 valuable positions to visit in alpha-beta pruning in order to save time on visiting nodes on the same level. So the searching level can be deeper.