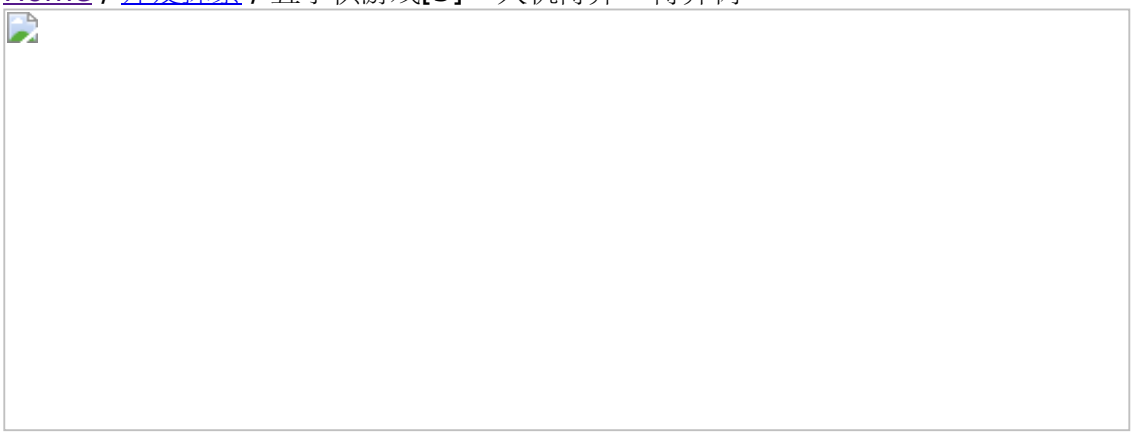
五子棋游戏[3] – 人机博弈 – 博弈树

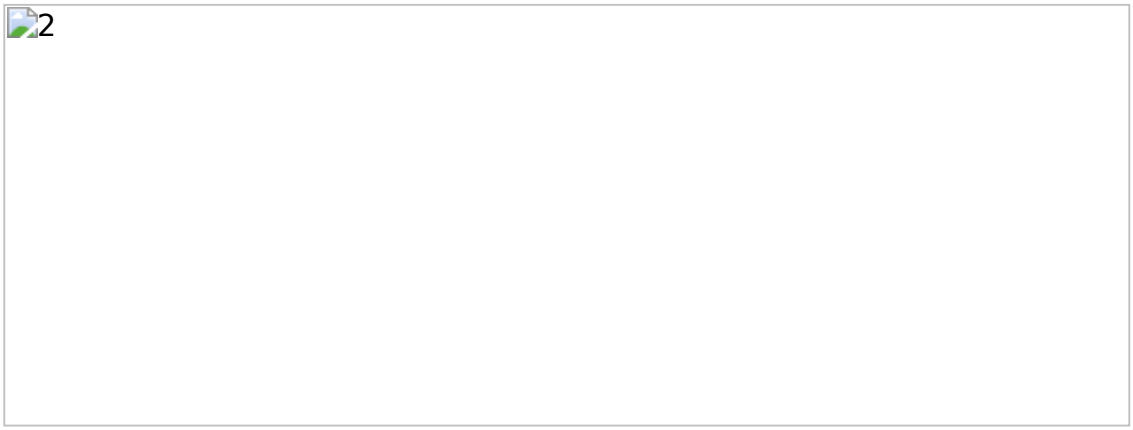
2013 年 07 月 17 日2014 年 08 月 07 日
Home / 开发探索 / 五子棋游戏[3] – 人机博弈 – 博弈树



距离上次写这个系列已经过去很久了，非常抱歉，现在呢我重新拾笔来填我自己挖下的这个坑。
在上一次的文章中，我们已经制作了整个游戏的最核心的框架，也就是说我们得到一个可以进行游戏的交互界面并且对游戏规则进行掌控的一个游戏类，在此前的文章中，我尽可能的让大家回避我所写的代码，因为这样会限制大家对整个游戏架构的思考，毕竟构建一个游戏的方式是多种多样的，我所想的方法并不一定是最好的，所以我希望大家自己去构建游戏的核心部分，我的文章仅作为一些提示。不过从这一篇文章开始，情况将有所转变，在完成的游戏核心部分之后，我希望让大家了解到的是，让游戏获得智能本身是一件又简单但又不简单的事情。
简单，是因为我们其实只是需要一个算法，能够替代两个玩家中的其中之一，这个算法的框架显而易见：


1

算法的**输入数据**就是当前棋局，算法通过分析棋局来计算并**得出**一个最佳的行棋位置。算法的需求是明朗简单的，但是难点就在于如何实现这个算法。
好在，有无数的人身先士卒，已经研究出了一类算法来应对这种博弈类问题，而且这种方法易于理解和实现，那就是博弈树的搜索。对于我们这种游戏，选用**极大极小博弈树(MGT)**是最为合适的，这种树用于描绘两个玩家交替进行游戏的过程，这里引用一张来自网络的图片*[1]*：


2

这是我们都非常熟悉的三子棋游戏，上图很清晰的展示了对局可能出现的所有情况（图的制作者已经去除了等价的情况），如果让这个图延展下去，我们就相当于穷举了所有的下法，如果我们能在知道所有下法的情况下，对这些下法加以判断，我们的AI自然就可以选择具有最高获胜可能的位置来下棋。
极大极小博弈树就是一种选择方法，由于五子棋以及大多数博弈类游戏是无法穷举出所有可能的步骤的（状态会随着博弈树的扩展而呈指数级增长），所以通常我们只会扩展有限的层数，而AI的智能高低，通常就会取决于能够扩展的层数，层数越高，AI了解的信息就越多，就越能做出有利于它的判断。

为了让计算机选择那些获胜可能性高的步骤走，我们就需要一个对局面进行打分的算法，越有利，算法给出的分数越高。在得到这个算法过后，计算机就可以进行选择了，在极大极小博弈树上的选择规则是这样的：

> AI会选择子树中具有最高估值叶子节点的路径；
> USER会选择子树中具有最小估值叶子节点的路径。

这样的原则很容易理解，作为玩家，我所选择的子一定要使自己的利益最大化，而相应的在考虑对手的时候，也不要低估他，一定要假设他会走对他自己最有利，也就是对我最不利的那一步。
这样的算法很容易实现，请参考下面的伪代码：

```
1  Function minmax(state, place):
2  newState = state.placePiece(place);
3  if(searchDepth > LIMIT_DEPTH) return evaluate(newState);
4  else
5  begin:
6  foreach(everyPossiblePlaces as nextPlace)
7  begin:
8  weight = minmax(newState, nextPlace);
9  if(min > weight) min = weight;
10 if(max < weight) max = weight;
11 end
12 end
13 Endfunction
```

由于我们通常设定的层数不会很多，所以我们使用递归来实现是最为直观方便的，每一层递归就是一个节点扩展其子节点的过程，并且我们要根据每层所代表的行棋方来选择最大或是最小的一个。
博弈树的构造是简单的，那困难的地方就是评价函数的构造，因为AI的智商除了取决于递归的层数外，最根本的，还是要看评价函数的好坏，那对于五子棋，我们又该如何去做呢？
要知道，直接分析整个棋面是一件很复杂的事情，为了让其具备可分析性，我们可以将其进行分解，分解成易于我们理解和实现的子问题。
对于一个二维的期面，五子棋不同于围棋，五子棋的胜负只取决于一条线上的棋子，所以根据五子棋的这一特征，我们就来考虑将二维的棋面转换为一维的，下面是一种简单的思考方式，对于整个棋盘，我们只需要考虑四个方向即可，所以我们就按照四个方向来将棋盘转换为15 * 4个长度不超过15的一维向量，参考下图：

根据这个图，我们来写分解棋盘的算法，下面不再是伪代码，而是C#代码，请仔细看：

```
1  /// <summary>
2  /// 评价一个棋面上的一方
3  /// </summary>
4  /// <param name="state">状态</param>
5  /// <param name="type">评价方</param>
6  /// <returns></returns>
7  public int evaluateState(ChessBoard state, int type)
8  {
9      int value = 0;
10     // 线状态
11     int[][] line = new int[6][];
12     line[0] = new int[17];
13     line[1] = new int[17];
14     line[2] = new int[17];
15     line[3] = new int[17];
16     line[4] = new int[17];
17     line[5] = new int[17];
18     int lineP;
19
20     // 方便检查边界
```

```
21    for (int p = 0; p < 6; ++p)
22        line[p][0] = line[p][16] = Globe.EVA_OP;
23
24    // 从四个方向产生
25    for (int i = 0; i < Globe.BOARD_SIZE; ++i)
26    {
27
28        // 产生线状态
29        lineP = 1;
30        for (int j = 0; j < Globe.BOARD_SIZE; ++j)
31        {
32            line[0][lineP] = Globe.getPieceType(state.chessBoard, i, j,type); // |
33            line[1][lineP] = Globe.getPieceType(state.chessBoard, j, i,type); // -
34            line[2][lineP] = Globe.getPieceType(state.chessBoard, i +j, j, type); // \
35            line[3][lineP] = Globe.getPieceType(state.chessBoard, i -j, j, type); // /
36            line[4][lineP] = Globe.getPieceType(state.chessBoard, j, i+ j, type); // \
37            line[5][lineP] = Globe.getPieceType(state.chessBoard,Globe.BOARD_SIZE - j - 1, i + j, type);
38 /
39            ++lineP;
40        }
41        // 评估线状态
42        int special = i == 0 ? 4 : 6;
43        for (int p = 0; p < special; ++p)
44        {
45            value += evaluateLine(line[p], true);
46        }
47    }
48
49 return value;
    }
```

唯一需要注意的一点就是，在分解斜向的时候，需要分为上下两个半区，所以我在分解的时候总共会有六个数组。
仅仅是分解为了线性状态还不够，我们的目的是为了为其评分，那么我们就还需要评估每个线状态，将每个线状态的评分进行汇总，当做我们的棋面评分：

进行到了这里，我们已经把二维的问题化简为了一维问题，那么接下来我们所要做的就是评价每一条线状态，根据五子棋的规则，我们可以很容易穷举出各种可能出现的基本棋型，我们首先为这些基本棋型进行识别和评价，并且统计每个线状态中出现了多少种下面所述的棋型，并据此得出评价值，那下表就是所谓的静态估值函数表：

根据这个表以及我们之前所谈到的规则，我们就可以得到下面的算法：

```
/// <summary>
/// 评价函数
/// 评价一行（一个方向）的棋子
/// 以center作为评估位置进行评价
/// </summary>
/// <param name="line">
/// 包含17个数字 最左边和最右边的数字为方便测试边界
/// -1 0 1 2 3 4 5 6 7 8 ...
///  X X X X X X X X X X
/// 代表待评价的一行棋子
/// 0 - 无子
/// 1 - 待评价方子
/// 2 - 对方子或无法下子
/// </param>
/// <returns></returns>
private int evaluateLine(int[] line, bool ALL)
{
    int value = 0; // 评估值
    int cnt = 0; // 连子数
    int blk = 0; // 封闭数

    // 从左向右扫描
```

```
23    for (int i = 0; i < Globe.BOARD_SIZE; ++i)
24    {
25        if (line[i] == Globe.EVA_MY) // 找到第一个己方的棋子
26        {
27            // 还原计数
28            cnt = 1;
29            blk = 0;
30            // 看左侧是否封闭
31            if (line[i - 1] == Globe.EVA_OP) ++blk;
32            // 计算连子数
33            for (i = i + 1; i < Globe.BOARD_SIZE && line[i] ==Globe.EVA_MY; ++i, ++cnt) ;
34            // 看右侧是否封闭
35            if (line[i] == Globe.EVA_OP) ++blk;
36            // 计算评估值
37            value += getValue(cnt, blk);
38        }
39    }
40    return value;
41 }
42
43 /// <summary>
44 /// 根据连字数和封堵数
45 /// 给出一个评价值
46 /// </summary>
47 /// <param name="cnt">连字数</param>
48 /// <param name="blk">封堵数</param>
49 /// <returns></returns>
50 private int getValue(int cnt, int blk)
51 {
52    if (blk == 0) // 活棋
53    {
54        switch (cnt)
55        {
56            case 1: return GameSetting.EVA_ONE;
57            case 2: return GameSetting.EVA_TWO;
58            case 3: return GameSetting.EVA_THREE;
59            case 4: return GameSetting.EVA_FOUR;
60            default: return GameSetting.EVA_FIVE;
61        }
62    }
63    else if (blk == 1) // 单向封死
64    {
65        switch (cnt)
66        {
67            case 1: return GameSetting.EVA_ONE_S;
```

```
68            case 2: return GameSetting.EVA_TWO_S;
69            case 3: return GameSetting.EVA_THREE_S;
70            case 4: return GameSetting.EVA_FOUR_S;
71            default: return GameSetting.EVA_FIVE;
72        }
73    }
74    else // 双向堵死
75    {
76        if (cnt >= 5) return GameSetting.EVA_FIVE;
77        else return GameSetting.EVA_ZERO;
78 }
79 }
```

通过这个算法，我们就可以解决整个游戏的局面评价问题，结合之前已经阐明了的博弈树算法，我们的五子棋游戏的AI，就已经具备智能了！如果你进行了实践，那么这时候，你就可以和自己编写的AI来进行对弈了~
那么本篇文章就到此结束了，下一次的博客将会讲解如何优化博弈树搜索的过程，使我们的AI在智商不变的情况下思考得更快：）