AI 简要说明

周宇星
zyx.pulsars@gmail.com

这是一个非常简单的 AI，基于这个框架，还可以进行大量改进以提高智商，代码已经进行了很基础的封装，关键过程的说明见注释。

以下可参考
https://en.wikipedia.org/wiki/Minimax
https://cs.uwaterloo.ca/~cbright/reports/cs686project.pdf，(Negamax 是 miniMax 的变体）

Minimax 算法

首先，同化棋是一种“0 和”对抗游戏（一方优势一方必然劣势）
那么我们考虑用通过 Minimax 算法解决。

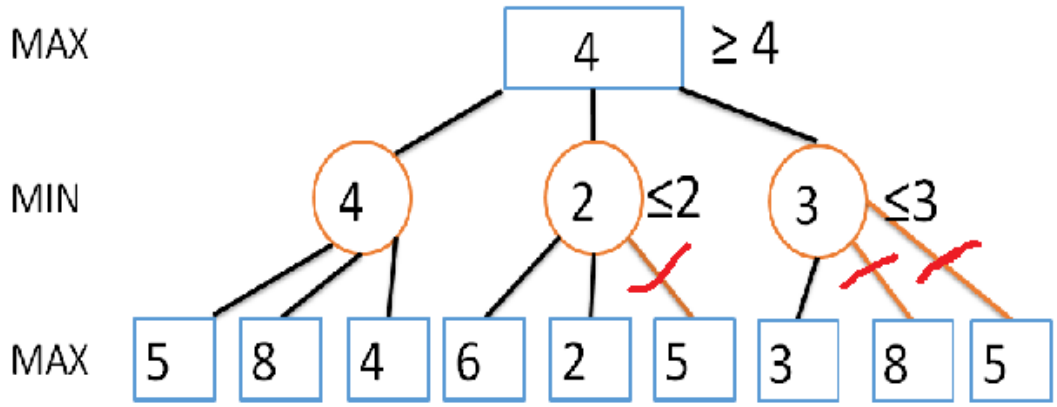
Minimax 算法又名极小化极大算法，是一种找出失败的最大可能性中的最小值的算法。

简单来说，就是枚举当前执行者所有可能的选择，不断一层层拓展局面，达到限定层数后，对局面进行估值，然后按 MAX 层选择分支的最大值，MIN 层选择分支的最小值，模拟对弈双方的选择。

α - β 剪枝

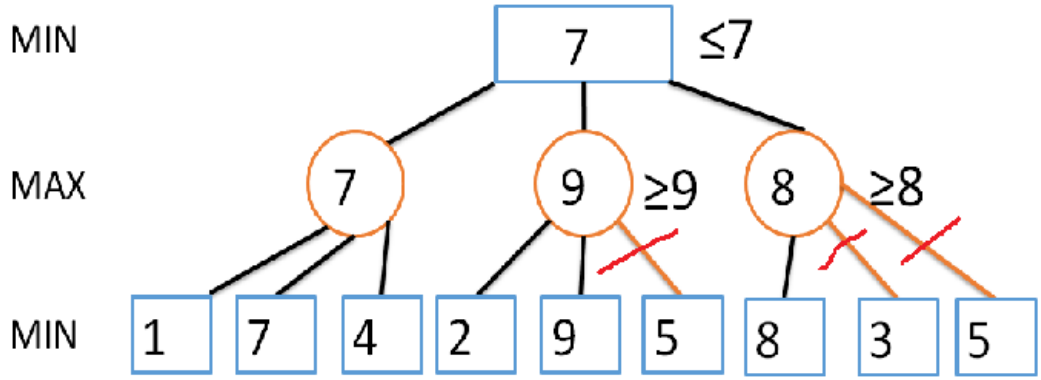
通过分析可以发现，在利用穷举方法执行 Minimax 算法中有许多的无效搜索，也就是说，许多明显较劣的状态分支我们也进行搜索了。

我们在进行极大值搜索的时候，我们仅仅关心，下面最大的状态，对于任何小于目前值的分支也都是完全没有必要进行**进一步**检查的。



（图中 MIN 层第二个结点，在得到下一层的估值 2 后，它的返回值必然<=2,而上一层是 MAX 层，从 MIN 层第一个结点已经得到了 4，故其>=4，所以，MIN 层第二个结点必然不会被选择，没有继续的必要）

同理，我们在进行极小值搜索的时候，我们仅仅关心，下面最小的状态，对于任何大于目前值的分支都是完全没有必要进行进一步检查的。



估价函数

使用 Minimax 算法，最为关键的部分是如何对一个局面进行估价，判断其到底有多大的可能性会给 AI 带来胜利。

分析同化棋的规则，容易发现其局面上的总棋子数是不减的。那么，双方获得的棋子的差是最容易想到的估价，也是此 AI 所使用的主要估价。

其次，经过尝试，当一方占据半壁江山（>24）后，非常容易走向胜利，所以我们贪心地尽量让其超过一半，即给予较高估价。

此 AI 可以进一步改进的地方：

1. 估价函数

   由于同化棋一步可以最多同化 8 个棋子，所以可能当前你优势（棋子差额）很大，但是下一步立马被翻盘。所以我们可以考虑统计一些容易被同化的如中空的方形，菱形的个数，给予估价值。

   还有很多估值方法…

2. 优化程序以加速

   根据 Minimax 算法的特点，在时限内搜索的层数越深，结点越多，选择就越优，AI 的智商就越高。此 AI 以易读性为主，牺牲了许多性能，可以做许多改进：

   1） 状态压缩

   对于一个棋盘，由于其只有 49 个位置，我们可以把位置映射到 0~48 这 49 个数（f(x,y) = x * 7 + y），对于某个位置，其只有存在棋子，不存在棋子，两种状态，所以对于对弈的一方，我们可以用一个 64 位整数的第 k 个 2 进制位=0/1 表示是否在 k 位置有这个数，从而把他的所有棋子的位置映射成一个 64 位整数（c++： long long int）

   这样一来，棋盘状态由一个 7*7 数组变成了两个整数，我们可以使用**位运算**，十分快速地进行查找位置，更新状态，统计个数等等操作。

   将压缩的状态 hash 判断冲突是防止进入循环局面的方法之一。

   2） 剪枝及搜索顺序

   如已经被已方及边界与最近对方分隔 2 层以上的己方完全占领区域的棋子，是可以稍后考虑的。

   接近对方边缘部位的棋子是可以优先考虑的，这样可以容易得出接近最优值的选择，更高效地利用 α - β 剪枝。

   3） 常数优化

根据语言及计算机的一些特性，在不改变功能的情况下对程序的语句进行修改，以期获得更好的性能。

…

3. 随机化

   可以用随机化在一定程度上防止局面陷入循环，或者被对手用同样的走法再次打败。但要注意，不能随机进入一个劣势局面。

4. 迭代加深？

   因为此 AI 现有估价及搜法较弱，最大搜索层数很低，所以在搜索时采用迭代加深层数限制的方法，这就产生了问题，下一个深度是沿用之前所有搜索深度得到的首层 max,min 值（通常在低深度得到最优值），还是初始化 max,min 值，现版本 AI 经测试，第一种占优，但是改进后，是否还进行迭代加深，或者采取哪种方法，需测试。（理论上第二种占优，本质上第一种是贪心）

…