五子棋AI报告

计算机1606班 20164625 戚子强

目录

[五子棋AI报告 1](#_Toc13389501)

[一、基本棋型 1](#_Toc13389502)

[二、赋权机制 4](#_Toc13389503)

[2.1 打分思路 4](#_Toc13389504)

[2.2 实现方法 5](#_Toc13389505)

[三、AI实现 5](#_Toc13389506)

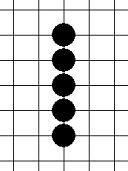
[3.1 基本算法 5](#_Toc13389507)

[3.2 性能分析 6](#_Toc13389508)

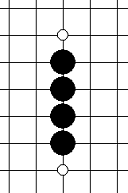
# 一、基本棋型

最常见的五子棋基本棋型大体有以下几种：**连五，活四，冲四，活三，眠三，活二，眠二**。

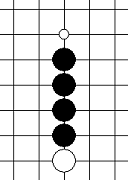
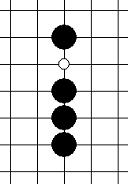
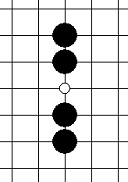
**①连五**：顾名思义，五颗同色棋子连在一起。



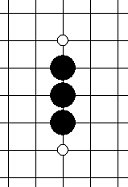
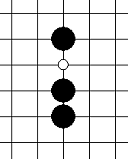
**②活四**：有两个连五点（即有两个点可以形成五），图中白点即为连五点。活四出现的时候，如果对方单纯过来防守的话，是已经无法阻止自己连五了。



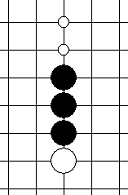
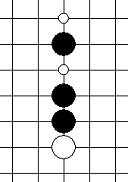
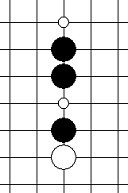
**③冲四**：有一个连五点，如下面三图，均为冲四棋型。图中白点为连五点。相对比活四来说，冲四的威胁性就小了很多，因为这个时候，对方只要跟着防守在那个唯一的连五点上，冲四就没法形成连五。

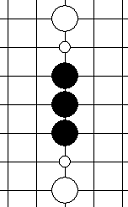
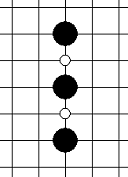
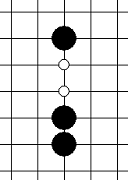
       

**④活三**：可以形成活四的三，如下图，代表两种最基本的活三棋型。图中白点为活四点。活三棋型是我们进攻中最常见的一种，因为活三之后，如果对方不以理会，将可以下一手将活三变成活四，而我们知道活四是已经无法单纯防守住了。所以，当我们面对活三的时候，需要非常谨慎对待。在自己没有更好的进攻手段的情况下，需要对其进行防守，以防止其形成可怕的活四棋型。其中，中间跳着一格的活三，也可以叫做**跳活三**。

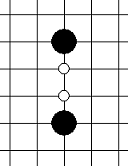
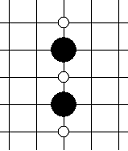
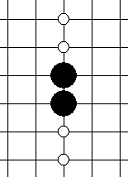
   

**⑤眠三：**只能够形成冲四的三，如下各图，分别代表最基础的六种眠三形状。图中白点代表冲四点。眠三的棋型与活三的棋型相比，危险系数下降不少，因为眠三棋型即使不去防守，下一手它也只能形成冲四，而对于单纯的冲四棋型，我们知道，是可以防守住的。

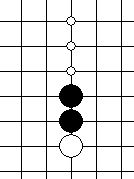
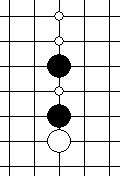
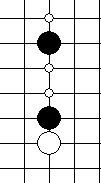
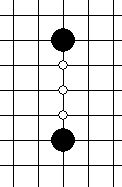
      



如上所示，眠三的形状是很丰富的。在下棋过程中，很容易忽略不常见的眠三形状。此外，在五子棋中，活四棋型比冲四棋型具有更大的优势，所以，我们在既能够形成活四又能够形成冲四时，会选择形成活四。学会判断一个三到底是活三还是眠三是非常重要的。后边禁手判断的时候也会有所应用。   
**⑥活二**：能够形成活三的二，如下图，是三种基本的活二棋型。图中白点为活三点。  
活二棋型看起来似乎很无害，因为他下一手棋才能形成活三，等形成活三，再防守也不迟。但其实活二棋型是非常重要的，尤其是在开局阶段，我们形成较多的活二棋型的话，当我们将活二变成活三时，才能够令自己的活三绵绵不绝微风里，让对手防不胜防。



**⑦眠二**：能够形成眠三的二。图中四个为最基本的眠二棋型，细心且喜欢思考的同学会根据眠三介绍中的图2-13找到与下列四个基本眠二棋型都不一样的眠二。图中白点为眠三点。

# 二、赋权机制

## 2.1 博弈树极大极小值alpha-beta剪枝搜索

博弈就是相互采取最优策略斗争的意思。比如下五子棋，假设棋盘的大小是10\*10，那就是100个点可以下，那么第一步可选择的可能就是100， 假设是下在了A点， 那么第二步就有除了A点的剩下的99个点的可能。 假设下在了B点， 那么第二步就有除了B点的剩下的99个点的可能，假设下在了C点...

如下图所示：

图片包含 文字, 地图

描述已自动生成

α为已知的最大值， β为已知的最小值， 因为还没搜索不知道是多少，保险起见，初始化为-∞ 和+∞。搜索到D的时候，局面得分是5，（深度优先搜索）那么也就是说要搜索最大值，那么只可能会在（5，+∞） 之间， 同理，要搜索最小值，也只会在（-∞，5）之间。 继续搜索， 搜索到G时，F暂时等于6 ，F是要找最大值， 那么F不可能再小于6了， 而B是要找最小值的，B的已知最小值是在（-∞，5）之间的， F无需搜索直接剪掉，同样对于另外一边的已知可能的极限范围β也是一样的情况，遇到就算是搜索也是浪费时间的情况，就剪枝不搜索了。 这样就减少了很多不必要是搜索步骤， 特别是一开始就找到最有可能的极大极小值， 更能迅速的剪枝。其伪代码表示如下：

int negamax(GameState S, int depth, int alpha, int beta) {

// 游戏是否结束 || 探索的递归深度是否到边界

if ( gameover(S) || depth == 0 ) {

return evaluation(S);

}

// 遍历每一个候选步

foreach ( move in candidate list ) {

S' = makemove(S);

value = -negamax(S', depth - 1, -beta, -alpha);

unmakemove(S')

if ( value > alpha ) {

// alpha + beta剪枝点

if ( value >= beta ) {

return beta;

}

alpha = value;

}

}

return alpha;

}

## 2.2 打分思路

（1）先对整个棋盘形势进行打分，存在两个矩阵（二维数组）上；

（2）一个为我方的形势分数，一个为敌方的形势分数；

（3）找出我方形势分数的最大值mymaxscore及其对应的位置，找出敌方形势的最大值hismaxscore及其对应的位置；

（4）判断是进攻还是防守：

如果mymaxscore>=hismaxscore，则进攻，下我方形势最大值mymaxscore对应的位置；如果有多个mymaxscore相等，则下这几个对应位置上hismaxscore最大的位置。

否则，防守，下敌方形势最大值hismaxscore对应的位置。如果有多个hismaxscore相等，则下这几个对应位置上mymaxscore最大的位置。

## 2.3 实现方法

（1）在棋盘空位置上预添加要判断放的棋子；

（2）取出以空位置为中心的4个方向（上，下，左，右），每个方向以该位置为中心两边各取4个格子信息。中心位置都是预放置，当前判断的时候位置还是空的；

（3）四个方向都判断其棋型，是否连五，活四，冲四，活三，眠三，活二，眠二等中的一种；

（4）最后综合四个方向的棋型，对该位置进行赋权，赋权分数如下：

r\_scoreLow2 = 20 # 眠二权值

r\_scoreHigh2 = 40 # 活二权值

r\_scoreLow3 = 60 # 眠三权值

r\_scoreHigh3 = 80 # 活三权值

r\_scoreLow4 = 100 # 冲四权值

r\_scoreHigh4 = 120 # 活四权值

r\_scoreHigh5 = 140 # 完成权值

u\_scoreLow2 = 10 # 眠二权值

u\_scoreHigh2 = 30 # 活二权值

u\_scoreLow3 = 50 # 眠三权值

u\_scoreHigh3 = 70 # 活三权值

u\_scoreLow4 = 90 # 冲四权值

u\_scoreHigh4 = 110 # 活四权值

u\_scoreHigh5 = 130 # 完成权值

# 三、AI实现

## 3.1 基本算法

（1）输入list\_me/list\_eney/list\_all，判断是否出现胜负，是则结束，否则继续；

（2）分为横、竖、左斜、右斜四个方向扫描，根据棋型和位置，进行初步剪枝；

（3）对剩余的位置进行分组对比进行组间的剪枝策略，做二次剪枝；

（4）对剩余的比较好的位置计算权值，取权值最大的位置记录到list；

（5）在权值相同情况下，找到list中最靠近上一次对手所下位置的位置，即为此次AI所下棋；

伪代码描述：

def compute\_X1(self, row, col): # X1=row,col,oleft,oright

for i in range(0, 15 - self.linkNum):

for m in range(self.linkNum, 15):

if(方向相同)

for k in range(self.linkNum):

# 遍历寻找棋型，并根据棋型分类

## 3.2 性能分析

实际调用下棋，笔记本配置为Ryzen R5 2500U + 8GB，Python 3.7.3情况下，AI出棋时间总可在1.8s之内。

但针对五子棋的特点，可以加一些其他优化，如搜索的开始点， 从上一步的点的周围搜索起，能尽快的找到相对较大的极大值，和相对较小的极小值，从而更快的剪枝。当然，棋型是很关键的一点，不仅仅用来打分，更可以加快剪枝速度。