EvaluationFunction介绍

参数：

board，当前棋盘结构，-1:空 0:己方 1:敌方，如下面所示。

board[i][j]表示i行j列。

返回值：

value，对当前棋盘局势的评估，大于0表示胜算大。

def evaluationFunction(*board*):

*# board = [[-1, 0, 1, 1, -1, -1, -1],*

*#          [-1, 0, 0, 1, -1, -1, -1],*

*#          [-1, 0, 0, 0,  1, -1, -1],*

*#          [-1, 1, 1, 0,  1, -1, -1],*

*#          [-1, 0, 1, 1,  1, -1, -1],*

*#          [-1, 1, 1, 0,  0,  0,  0]]*

常数：

COLUMN列数，ROW行数。

FOUR：连成四子的棋形的权值。

DEAD：一侧为敌方棋子，一侧为空。

ALIVE：两侧均为空。

相同连子数量的棋形，ALIVE权值大于DEAD。

*#constant*

    COLUMN = 7

    ROW = 6

    FOUR = 2000000

    ALIVE\_3 =  100000

    DEAD\_3 = 1000

    ALIVE\_2 = 800

    DEAD\_2 = 5

    ALIVE\_1 = 3

    DEAD\_1 = 1

变量：

value，表示待返回的权值。

cal\_board:board的拷贝。

cal\_top:每列实际列顶，列空：cal\_top[i] = ROW,列满:cal\_top[i] = 0。

d:存储FOUR和DEAD棋形的数量。第一维：棋形。第二维：玩家和机器。

a:存储ALIVE棋形的数量。第一维：棋形。第二维：玩家和机器。

cons:活眼的加权值。

可以这样理解：每一种棋形，想要延长它，必须将棋子下到他的活眼上，而活眼底下的空白处越多，越不容易将棋子下到该活眼，也就越难延长该棋形，该棋形的价值相应下降。

活眼下无空白处对应cons[0]=1,活眼下有i个空白对应cons[i]=0.5i,i小于等于5。

*#variables*

    value = 0.0

    cal\_board = board

cal\_top = [] *#每列实际列顶，列空：cal\_top[i] = ROW,列满:cal\_top[i] = 0*

*#计算cal\_top*

*for* j in range(COLUMN):

*for* i in range(ROW):

*if* cal\_board[i][j] != -1:

                cal\_top.append(i)

*break*

*if* i == ROW-1:

                cal\_top.append(ROW)

    print(cal\_top)

*#第一维：棋形。第二维：玩家和机器人*

    d = [[0]\*2 *for* i in range(5)]*#d[5][2]*

    a = [[0]\*2 *for* i in range(4)]*#a[4][2]*

*#活眼加权值*

    cons = [1.0, 0.5, 0.25, 0.125, 0.0625, 0.03125]

方法：

cal:

Player的取值为0或1，若为0，则表示此时计算己方各棋形数量，若为1，则表示此时计算对方各棋形数量。

分别考虑四个方向上的可能，具体代码详见源代码。

*#methods*

def cal(*player*):*#type为0或1，表示计算的是玩家或机器人的各棋形数*

*#横向*

*#纵向*

*#左下-右上*

*#左上-右下*

主要部分：

该部分即计算value的值

*#main*

    cal(0)

    value+=d[4][0]\*FOUR+d[3][0]\*DEAD\_3+d[2][0]\*DEAD\_2+d[1][0]\*DEAD\_1+a[3][0]\*ALIVE\_3+a[2][0]\*ALIVE\_2+a[1][0]\*ALIVE\_1

    cal(1)

    value-=d[4][1]\*FOUR+d[3][1]\*DEAD\_3+d[2][1]\*DEAD\_2+d[1][1]\*DEAD\_1+a[3][1]\*ALIVE\_3+a[2][1]\*ALIVE\_2+a[1][1]\*ALIVE\_1

*return* value