

AI黑白棋

中山大学计算机学院

人工智能

本科生实验报告

(2022学年春季学期)

课程名称：Artificial Intelligence

教学班级	冯班	专业（方向）	计算机科学与技术（系统结构）
学号	20337188	姓名	邓理华

一、 实验题目

- 实现8×8的黑白翻转棋的人机对战：
 - 要求使用alpha-beta剪枝
 - 不要求实现UI
 - 搜索深度和评价函数不限，自己设计。在报告中说明清楚自己的评价函数及搜索策略
 - 鼓励大家结合高级搜索算法优化评价函数
 - 实验结果要求展示至少连续三个回合（人和机器各落子一次指一回合）的棋局分布情况，要求输出每步得分

二、 实验内容

算法原理

搜索策略

Minimax算法

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。

Minimax算法常用于棋类等由两方较量的游戏和程序。该算法是一个零总和算法，即一方要在可选的选项中选择将其优势最大化的选择，另一方则选择令对手优势最小化的方法。而开始的时候总和为0。

AlphaBeta剪枝

Alpha-beta($\alpha - \beta$)剪枝的名称来自计算过程中传递的两个边界，这些边界基于已经看到的搜索树部分来限制可能的解决方案集。其中，Alpha(α)表示目前所有可能解中的最大下界，Beta(β)表示目前所有可能解中的最小上界。

因此，如果搜索树上的一个节点被考虑作为最优解的路上的节点（或者说是这个节点被认为是有必要进行搜索的节点），那么它一定满足以下条件（N是当前节点的估价值）：

$$\alpha \leq N \leq \beta$$

在我们进行求解的过程中， α 和 β 会逐渐逼近。如果对于某一个节点，出现了 $\alpha > \beta$ 的情况，那么，说明这个点一定不会产生最优解了，所以，我们就不再对其进行扩展（也就是不再生成子节点），这样就完成了对博弈树的剪枝。

评价函数

(1) $f(\text{局面}) = \text{子力(我的子数} - \text{对手的子数)}$

(2) $g(\text{局面}) = \text{角(我控制的} - \text{对手控制的)}$

(3) $h(\text{局面}) = \text{机动性(我可以走的)}$

组合这些函数来构成一个评价函数：

eval = gapWeight·f(局面) + cornerWeight·g(局面) + mobilityWeight·h(局面)

评价函数的优化：模拟退火算法

模拟退火算法借鉴了统计物理学的思想，是一种简单、通用的启发式优化算法，并在理论上具有概率性全局优化性能，因而在科研和工程中得到了广泛的应用。

退火是金属从熔融状态缓慢冷却、最终达到能量最低的平衡态的过程。模拟退火算法基于优化问题求解过程与金属退火过程的相似性，以优化目标为能量函数，以解空间为状态空间，以随机扰动模拟粒子的热运动来求解优化问题。

模拟退火算法结构简单，由温度更新函数、状态产生函数、状态接受函数和内循环、外循环终止准则构成。

模拟退火算法的基本流程如下：

- (1) 初始化：初始温度T，初始解状态s，迭代次数L；
- (2) 对每个温度状态，重复 L 次循环产生和概率性接受新解；
- (3) 通过变换操作由当前解s 产生新解s'；
- (4) 计算能量差 ΔE ，即新解的目标函数与原有解的目标函数的差；
- (5) 若 $\Delta E < 0$ 则接受s'作为新的当前解，否则以概率 $\exp(-\Delta E/T)$ 接受s' 作为新的当前解；
- (6) 在每个温度状态完成 L 次内循环后，降低温度 T，直到达到终止温度。

关键代码展示（带注释）

grid.py 棋盘设计

```
import pysnooper
from copy import deepcopy
class Grid(object):
    def __init__(self):
        self.mygrid=[ ['*' for row in range(8)] for col in range(8)]
        self.mygrid[3][3], self.mygrid[4][4]='o', 'o'
        self.mygrid[3][4], self.mygrid[4][3]='x', 'x'
        self.empty='*'

    def count(self, type):
        num=0
        for row in range(8):
            for col in range(8):
                if self.mygrid[row][col]==type:
                    num+=1
        return num

    def __getitem__(self, index):
        return self.mygrid[index]
    def countBoth(self):
        black_num=0
        white_num=0
        for row in range(8):
```

```

        for col in range(8):
            if self.mygrid[row][col]=='X':
                black_num+=1
            if self.mygrid[row][col]=='O':
                white_num+=1
        return black_num,white_num

def countCorners(self,type):
    cnt=0
    grid=self.mygrid
    if grid[0][0]==type:
        cnt += 1
    if grid[0][7]==type:
        cnt += 1
    if grid[7][7]==type:
        cnt += 1
    if grid[7][0]==type:
        cnt += 1
    return cnt

def see(self):
    grid = self.mygrid
    print("棋局分布情况: ")
    print("R\c A B C D E F G H")
    for row in range(8):
        print(str(row+1), ' ', ' '.join(grid[row]))

def whowon(self):
    black_count, white_count = 0, 0
    for i in range(8):
        for j in range(8):
            if self.mygrid[i][j] == 'X':
                black_count += 1
            if self.mygrid[i][j] == 'O':
                white_count += 1
    if black_count > white_count:
        # 黑棋胜
        return 0, black_count - white_count
    elif black_count < white_count:
        # 白棋胜
        return 1, white_count - black_count
    elif black_count == white_count:
        # 表示平局，黑棋个数和白旗个数相等
        return 2, 0

#@pysnooper.snoop("C:/Users/Administrator/Desktop/ai_lab2_log/debug.log",
prefix="--*--")

def NextStatus(self, action, type):
    # 判断action 是不是字符串, 如果是则转化为数字坐标
    if isinstance(action, str):
        action = self.StrToArray(action)

    fliped = self.GetReverseGrid(action, type)

    if fliped:
        for flip in fliped:
            x, y = self.StrToArray(flip)
            self.mygrid[x][y] = type

```

```

# 落子坐标
x, y = action
# 更改棋盘上 action 坐标处的状态，修改之后该位置属于 type[X,O,.]等三状态
self.mygrid[x][y] = type
return fliped

else:
    # 没有反转子则落子失败
    return False

def iswithinRange(self, x, y):
    return x >= 0 and x <= 7 and y >= 0 and y <= 7

def StrToArray(self, position):
    row='12345678'.index(str(position[1]))
    col='ABCDEFGH'.index(str(position[0]))
    return row, col

#输入元组
def ArrayToStr(self, act_tuple):
    row,col=act_tuple
    return chr( ord('A')+col )+str(row+1)

def BP(self, action, flipped_pos, type):
    # 判断action 是不是字符串，如果是则转化为数字坐标
    if isinstance(action, str):
        action = self.StrToArray(action)

    self.mygrid[action[0]][action[1]] = self.empty
    # 如果 type == 'X'，则 op_type = 'O';否则 op_type = 'X'
    op_type = "O" if type == "X" else "X"

    for p in flipped_pos:
        if isinstance(p, str):
            p = self.StrToArray(p)
        self.mygrid[p[0]][p[1]] = op_type

def GetReverseGrid(self, action, type):
    if isinstance(action, str):
        action = self.StrToArray(action)
    xstart, ystart = action

    # 如果该位置已经有棋子或者出界，返回 False
    if not self.iswithinRange(xstart, ystart) or self.mygrid[xstart][ystart] != self.empty:
        return False

    # 临时将type放到指定位置
    self.mygrid[xstart][ystart] = type
    # 棋手
    op_type = "O" if type == "X" else "X"

    # 要被翻转的棋子
    flipped_pos = []
    flipped_pos_board = []

    for xdirection, ydirection in [[0, 1], [1, 1], [1, 0], [1, -1], [0, -1],
[-1, -1], [-1, 0],
```

```

[-1, 1]]:

x, y = xstart, ystart
x += xdirection
y += ydirection
# 如果(x,y)在棋盘上, 而且为对方棋子, 则在这个方向上继续前进, 否则循环下一个角度。
if self.iswithinRange(x, y) and self.mygrid[x][y] == op_type:
    x += xdirection
    y += ydirection
    # 进一步判断点(x,y)是否在棋盘上, 如果不在棋盘上, 继续循环下一个角度, 如果在棋
    盘上, 则进行while循环。
    if not self.iswithinRange(x, y):
        continue
    # 一直走到出界或不是对方棋子的位置
    while self.mygrid[x][y] == op_type:
        # 如果一直是对方的棋子, 则点(x,y)一直循环, 直至点(x,y)出界或者不是对
        方的棋子。
        x += xdirection
        y += ydirection
        # 点(x,y)出界了和不是对方棋子
        if not self.iswithinRange(x, y):
            break
        # 出界了, 则没有棋子要翻转XXXXXX
        if not self.iswithinRange(x, y):
            continue

        # 是自己的棋子XXXXXXXXO
        if self.mygrid[x][y] == type:
            while True:
                x -= xdirection
                y -= ydirection
                # 回到了起点则结束
                if x == xstart and y == ystart:
                    break
                # 需要翻转的棋子
                flipped_pos.append([x, y])

# 将前面临时放上的棋子去掉, 即还原棋盘
self.mygrid[xstart][ystart] = self.empty # restore the empty space

# 没有要被翻转的棋子, 则走法非法。返回 False
if len(flipped_pos) == 0:
    return False

for fp in flipped_pos:
    flipped_pos_board.append(self.ArrayToStr(fp))
# 走法正常, 返回翻转棋子的棋盘坐标
return flipped_pos_board

def GetLegalPos(self, type):
    direction = [(-1, 0), (-1, 1), (0, 1), (1, 1), (1, 0), (1, -1), (0, -1),
    (-1, -1)]

    op_type = "O" if type == "X" else "X"
    # 统计 op_type 一方邻近的未落子状态的位置
    op_type_near_points = []

    grid = self.mygrid
    for i in range(8):

```

```

# i 是行数, 从0开始, j是列数, 也是从0开始
for j in range(8):
    # 判断棋盘[i][j]位子棋子的属性, 如果是op_type, 则继续进行下一步操作,
    # 否则继续循环获取下一个坐标棋子的属性
    if grid[i][j] == op_type:
        # dx, dy 分别表示[i][j]坐标在行、列方向上的步长, direction 表示方向
        for dx, dy in direction:
            x, y = i + dx, j + dy
            # 表示x、y坐标值在合理范围, 棋盘坐标点board[x][y]为未落子状态,
            # 而且(x,y)不在op_type_near_points 中, 统计对方未落子状态位置
            的列表才可以添加该坐标点
            if 0 <= x <= 7 and 0 <= y <= 7 and grid[x][y] ==
self.empty and (
                x, y) not in op_type_near_points:
                op_type_near_points.append((x, y))
    l = [0, 1, 2, 3, 4, 5, 6, 7]
legalList=[]
for p in op_type_near_points:
    if self.GetReverseGrid(p, type):
        if p[0] in l and p[1] in l:
            p = self.ArrayToStr(p)
    legalList.append(p)
return legalList

```

play.py 实现人机对弈

```

from func_timeout import func_timeout, FunctionTimedOut
import datetime
from grid import Grid
from copy import deepcopy
INF=9999999

class Me:
    def __init__(self, type):
        self.type = type

    def GetNextStatus(self, grid):
        if self.type == "X":
            gamer = "黑棋"
        else:
            gamer = "白棋"
        while True:
            action = input(
                "请{}-{}'输入: ".format(gamer, self.type))

            if action == "Q" or action == 'q':
                return "Q"
            else:
                row, col = action[1].upper(), action[0].upper()

                if row in '12345678' and col in 'ABCDEFGH':
                    if action in grid.GetLegalPos(self.type):
                        return action
                else:
                    print("重新输入")

```

```

class AI:
    d = 0
    MAXd = 3
    def __init__(self,type,gapW,cornerW,mobilityW):
        self.gapW, self.cornerW, self.mobilityW=gapW,cornerW,mobilityW
        self.type=type

    def setweight(self,gapW, cornerW, mobilityW):
        self.gapW, self.cornerW, self.mobilityW=gapW, cornerW, mobilityW

    def evaluate(self,grid,type):
        """
        (1) f(局面) = 子力(我的子数 - 对手的子数)
        (2) g(局面) = 角(我控制的 - 对手控制的)
        (3) h(局面) = 机动性(我可以走的)
        组合这些函数来构成一个评价函数: eval = gapWeight·f + cornerWeight·g +
        mobilityWeight·h
        """

        black_num, white_num=grid.countBoth()
        if type == 'X':
            gap=black_num-white_num
        else:
            gap=white_num-black_num
        value=int(self.gapW*gap +self.cornerW*grid.countCorners(type)
        +self.mobilityW*len( list(grid.GetLegalPos(type)) ))
        return value

    def miniMax(self,grid,type,a,b):
        if self.d > self.MAXd: #end
            if type == self.type:
                return None, self.evaluate(grid,type)
            else:
                return None, -self.evaluate(grid,type)

        if type == 'X':
            typeNext ='O'
        else:
            typeNext = 'X'
        action_list = list(grid.GetLegalPos(type))
        if len(action_list) == 0:
            if len(list(grid.GetLegalPos(typeNext))) == 0:
                return None,self.evaluate(grid,type)
            return self.miniMax(grid,typeNext,a,b)

        max = -INF
        min = INF
        action = None

        for p in action_list:

            flipped_pos = grid.NextStatus(p,type)
            self.d += 1
            pl, current = self.miniMax(grid,typeNext,a,b)
            self.d -= 1
            grid.BP(p,flipped_pos,type)

            # alpha-beta 剪枝

```

```

        if type == self.type:
            if current > a:
                if current > b:
                    return p,current
                a = current
            if current > max:
                max = current
                action = p

        else:
            if current < b:
                if current < a:
                    return p,current
                b = current
            if current < min:
                min = current
                action = p
    if type == self.type:
        return action,max
    else:
        return action,min

def GetNextStatus(self, grid):
    if self.type == 'X':
        gamer_name = '黑棋'
    else:
        gamer_name = '白棋'
    print("轮到 {}-{} 落子".format(gamer_name, self.type))
    action_list = list(grid.GetLegalPos(self.type))

    action, weight = self.miniMax(grid, self.type, -INF, INF)
    print("本次落子得分: ", weight)

    if len(action_list) == 0:
        return None
    print(action_list)
    print(action)
    return action

class Play(object):
    def __init__(self, blackGamer, whiteGamer):
        self.grid = Grid()
        self.current_gamer = None
        self.blackGamer = blackGamer
        self.whiteGamer = whiteGamer
        self.blackGamer.type = "X"
        self.whiteGamer.type = "O"

    def printWhoWon(self, winIndex):
        print(['胜者为黑棋', '胜者为白棋', '平局'][winIndex])

    def changeGamer(self, blackGamer, whiteGamer):
        if self.current_gamer is None:
            return blackGamer
        else:
            if self.current_gamer == self.blackGamer:
                return whiteGamer
            else:

```

```

        return blackGamer

def foul(self, is_timeout=False, is_board=False, is_legal=False):
    if self.current_gamer == self.blackGamer:
        win_type = '白棋 - O'
        loss_type = '黑棋 - X'
        winIndex = 1
    else:
        win_type = '黑棋 - X'
        loss_type = '白棋 - O'
        winIndex = 0

    if is_timeout:
        print('\n{} 超时, {} 胜'.format(loss_type, win_type))
    if is_legal:
        print('\n{} 落子 3 次不符合规则, 故 {} 胜'.format(loss_type, win_type))
    if is_board:
        print('\n{} 擅自改动棋盘判输, 故 {} 胜'.format(loss_type, win_type))

gap = 0

return winIndex, gap

def gameOver(self):
    b_list = list(self.grid.GetLegalPos('X'))
    w_list = list(self.grid.GetLegalPos('O'))

    overFlag = len(b_list) == 0 and len(w_list) == 0

    return overFlag

def playothello(self):
    # 初始化胜负结果和棋子差
    winIndex = None
    gap = -1

    # 游戏开始
    print('\n---Game Start\n')
    # 棋盘初始化
    self.grid.see()
    while True:
        self.current_gamer = self.changeGamer(self.blackGamer,
self.whiteGamer)
        start_time = datetime.datetime.now()
        type = "X" if self.current_gamer == self.blackGamer else "O"
        # 获取当前下棋方合法落子位置
        legal_actions = list(self.grid.GetLegalPos(type))
        if len(legal_actions) == 0:
            if self.gameOver():
                # 游戏结束, 双方都没有合法位置
                winIndex, gap = self.grid.whowon()
                break
            else:
                # 另一方有合法位置, 切换下棋方
                continue

grid = deepcopy(self.grid.mygrid)

```

```

# legal_actions 不等于 0 则表示当前下棋方有合法落子位置
try:
    for i in range(0, 3):
        # 获取落子位置
        action = func_timeout(60,
self.current_gamer.GetNextStatus, kwargs={'grid': self.grid})
        if action not in legal_actions:
            # 判断当前下棋方落子是否符合合法落子,如果不合法,则需要对方重新输入
            print("不合法, 请重新落子")
            continue
        else:
            # 落子合法则直接 break
            break
    else:
        # 落子3次不合法
        winIndex, gap = self.foul(is_legal=True)
        break
except FunctionTimedOut:
    # 落子超时, 结束游戏
    winIndex, gap = self.foul(is_timeout=True)
    break

# 结束时间
end_time = datetime.datetime.now()
if grid != self.grid.mygrid:
    # 修改棋盘, 结束游戏
    winIndex, gap = self.foul(is_board=True)
    break
if action is None:
    continue
else:
    # 统计一步所用的时间
    es_time = (end_time - start_time).seconds
    if es_time > 60:
        # 该步超过60秒则结束比赛。
        print('\n{} 超时'.format(self.current_gamer))
        winIndex, gap = self.foul(is_timeout=True)
        break

    # 当前玩家颜色, 更新棋局
    self.grid.NextStatus(action, type)
    # 显示当前棋盘
    self.grid.see()

    # 判断游戏是否结束
    if self.gameOver():
        # 游戏结束
        winIndex, gap = self.grid.whowon() # 得到赢家 0,1,2
        break

print('\n---End of the game\n')
self.grid.see()
self.printwhowon(winIndex)

if winIndex is not None and gap > -1:
    return winIndex

def main():

```

```

blackGamer = AI("X", 0.025044, 4.541892, 37.444960)
whiteGamer = Me("O")
play = Play(blackGamer, whiteGamer)
play.playOthello()

if __name__ == '__main__':
    main()

```

SA.py 模拟退火算法

```

from play import AI,Play
import math
import random
import numpy as np
from pickle import TRUE

def ParameterSetting():
    problem = "othello_opt"
    varNum = 3
    # 搜索空间上下限
    xMin = [0.01, 1, 10]
    xMax = [0.05, 5, 50]
    tInitial = 100.0
    tFinal = 60
    # 降温参数
    alpha = 0.8
    # 内循环运行次数
    innerLoopLen = 5
    # 搜索步长
    step = 0.5
    return problem, varNum, xMin, xMax, tInitial, tFinal, alpha, innerLoopLen,
           step

def calculate(xNew,xNow):
    solValue=0
    black_player=AI('X',xNew[0],xNew[1],xNew[2])
    white_player=AI('O',xNow[0],xNow[1],xNow[2])
    play=Play(black_player,white_player)
    result=play.playOthello()
    if result==0:
        #xNew更优
        solValue+=1
    return solValue

def SA(varNum,xMin,xMax,tInitial,tFinal,alpha,innerLoopLen,step):
    # 初始化开始
    # 创建数组
    xInitial = np.zeros((varNum))
    for v in range(varNum):
        xInitial[v] = random.uniform(xMin[v], xMax[v])
    xNew = np.zeros((varNum))
    xNow = np.zeros((varNum))
    xBest = np.zeros((varNum))
    xNow[:] = xInitial[:]
    xBest[:] = xInitial[:]
    print('x_Initial:{:.6f},{:.6f},\t'.format(xInitial[0], xInitial[1]))
    # 外循环次数

```

```

recordIter = []
# 当前解的目标函数值
recordxNow = []
# 最佳解的目标函数值
recordxBest = []
# 劣质解的接受概率
recordPBad = []
# 外循环迭代次数, 温度状态数
kIter = 0
# 总计内循环次数
innerIterSum = 0
# 内循环次数
nInnerIter = innerLoopLen
# 初始化结束

# 算法开始
# 外循环开始
tNow = tInitial
while tNow >= tFinal:
    # 获得优质解、接受优解、拒绝劣解的次数
    kBetter = 0
    kBadAccept = 0
    kBadRefuse = 0

    # 内循环开始
    for k in range(nInnerIter):
        innerIterSum += 1

        # 产生新解: 只改一个解
        xNew[:] = xNow[:]
        v = random.randint(0, varNum-1)
        xNew[v] = xNow[v] + step * (xMax[v]-xMin[v]) *
random.normalvariate(0, 1)
        # 保证新解范围内
        xNew[v] = max(min(xNew[v], xMax[v]), xMin[v])

        # 能量差简化为1
        deltaE = 1

        # 按 Metropolis 准则接受新解, 如果新解的目标函数好于当前解, 则接受新解
        if calculate(xNew, xNow):
            accept = True
            kBetter += 1
        else:
            pAccept = math.exp(-deltaE / tNow)
            if pAccept > random.random():
                accept = TRUE
                kBadAccept += 1
            else:
                accept = False
                kBadRefuse += 1

        # 保存新解
        # 如果接受新解, 则将新解保存为当前解
        if accept == True:
            xNow[:] = xNew[:]
            # 如果新解的目标函数好于最优解, 则将新解保存为最优解
            if calculate(xNew, xBest):

```

```

        xBest = xNew
        xBest[:] = xNew[:]
        # 可变搜索步长，逐步减小搜索范围，提高搜索精度
        step = step*0.99

        # 内循环结束，保存数据
        # 劣质解的接受概率
        pBadAccept = kBadAccept / (kBadAccept + kBadRefuse)
        # 当前外循环次数
        recordIter.append(kIter)
        recordxNow.append(xNow)
        recordxBest.append(xBest)
        #四位小数
        recordPBad.append(round(pBadAccept, 4))
        # 定时输出一下劣解接受概率
        if kIter%10 == 0:
            print('i:{} ,t(i):{:.2f}, badAccept:{:.6f}' .\
                  format(kIter, tNow, pBadAccept))

        # 降温
        tNow = tNow * alpha
        kIter = kIter + 1
    #外循环结束
#算法结束

return kIter,xBest,xNow,recordIter,recordxNow,recordxBest,recordPBad

# 结果校验与输出
def
ResultOutput(problem,varNum,xBest,kIter,recordxNow,recordxBest,recordPBad,record
Iter):
    print("\n优化结果:")
    for i in range(varNum):
        print('\tx[{}] = {:.6f}'.format(i,xBest[i]))

    return

def main():
    [problem, varNum, xMin, xMax, tInitial, tFinal, alpha, innerLoopLen, step] =
ParameterSetting()
    [kIter,xBest,xNow,recordIter,recordxNow,recordxBest,recordPBad] \
        = SA(varNum,xMin,xMax,tInitial,tFinal,alpha,innerLoopLen,step)
    ResultOutput(problem,
varNum,xBest,kIter,recordxNow,recordxBest,recordPBad,recordIter)

if __name__ == '__main__':
    main()

```

创新点

使用pysnooper打印日志调试

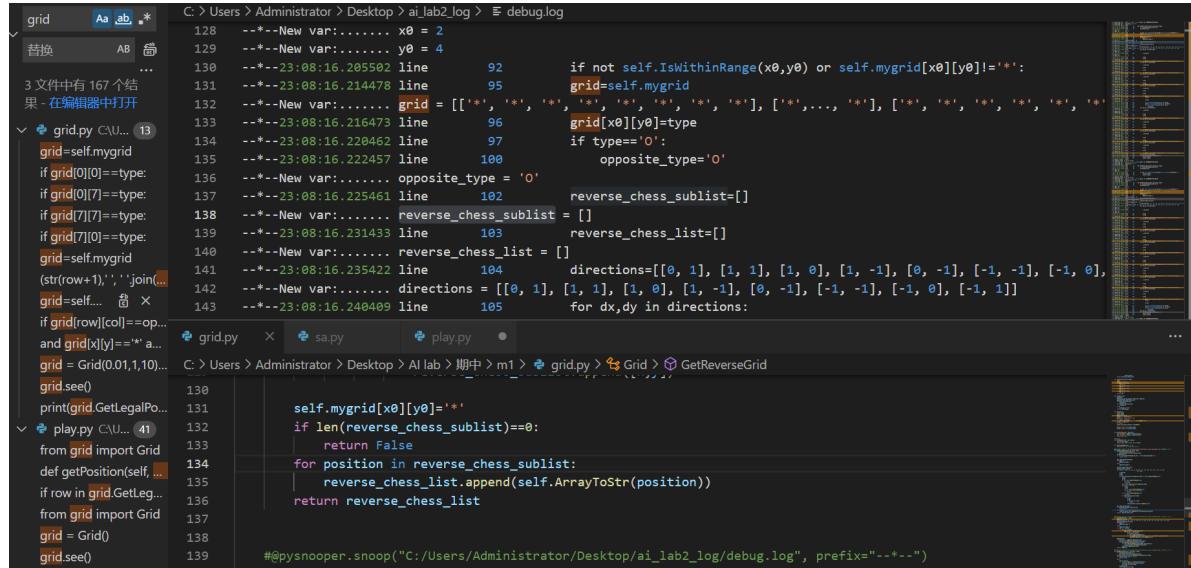
该工具使用采用装饰器的形式，将函数的运行过程以日志的形式打印到文件中，其记录了运行了哪些代码行，运行的时间及运行到当前代码时各变量的值。根据变量的变化就可以定位问题了。亲自试用该工具后，其优点可总结为以下几点：

- 1、无需为了查看变量的值，使用print打印变量的值，从而修改了原有的代码。

2、接口的运行过程以日志的形式保存，方便随时查看。

3、可以根据需要，设置函数调用的函数的层数，方便将注意力集中在需要重点关注的代码段。

4、多个函数的日志，可以设置日志前缀表示进行标识，方便查看时过滤。



```
C: > Users > Administrator > Desktop > ai_lab2_log > debug.log
128 ---*-New var:..... x0 = 2
129 ---*-New var:..... y0 = 4
130 ---*-23:08:16.205502 line 92 if not self.IsWithinRange(x0,y0) or self.mygrid[x0][y0]!='*':
131     grid=self.mygrid
132     ---*-New var:..... grid = [['*', '*', '*', '*', '*', '*', '*', '*'], ['*', ..., '**'], ['*', '**', '**', '**', '**'],
133     ---*-23:08:16.214478 line 95 grid[x0][y0]=type
134     ---*-23:08:16.216473 line 96 grid[x0][y0]=type
135     ---*-23:08:16.222457 line 100 if type=='O':
136         opposite_type = 'O'
137         ---*-23:08:16.225461 line 102 reverse_chess_sublist=[]
138         ---*-New var:..... reverse_chess_sublist = []
139         ---*-23:08:16.231433 line 103 reverse_chess_list=[]
140         ---*-New var:..... reverse_chess_list = []
141         ---*-23:08:16.235422 line 104 directions=[[0, 1], [1, 1], [1, 0], [1, -1], [0, -1], [-1, -1], [-1, 0], [-1, 1]]
142         ---*-New var:..... directions = [[0, 1], [1, 1], [1, 0], [1, -1], [0, -1], [-1, -1], [-1, 0], [-1, 1]]
143         ---*-23:08:16.240409 line 105 for dx,dy in directions:
144             ---*-23:08:16.240409 line 106 if len(reverse_chess_sublist)==0:
145                 return False
146                 for position in reverse_chess_sublist:
147                     reverse_chess_list.append(self.ArrayToStr(position))
148                 return reverse_chess_list
149
#pysnooper.snoop("C:/Users/Administrator/Desktop/ai_lab2_log/debug.log", prefix="---")
```

三、实验结果及分析

1 实验结果展示示例（可图可表可文字，尽量可视化）

```
PS C:\Users\Administrator> & C:/Users/Administrator/AppData/Local/Programs/Python/Python310/python.exe "c:/Users/Administrator/Desktop/AI lab/期中/v2/play.py"

---Game Start

棋局分布情况:
R\C A B C D E F G H
1 * * * * * * *
2 * * * * * * *
3 * * * * * * *
4 * * * O X * * *
5 * * * X O * * *
6 * * * * * * *
7 * * * * * * *
8 * * * * * * *

轮到 黑棋-X 落子
本次落子得分: 187
['D3', 'C4', 'F5', 'E6']
D3
棋局分布情况:
R\C A B C D E F G H
1 * * * * * * *
2 * * * * * * *
3 * * * X * * *
4 * * * X X * * *
5 * * * X O * * *
6 * * * * * * *
7 * * * * * * *
8 * * * * * * *

请'白棋-O'输入: E3
棋局分布情况:
```

```
R\C A B C D E F G H
1 * * * * * * * *
2 * * * * * * * *
3 * * * X O * * *
4 * * * X O * * *
5 * * * X O * * *
6 * * * * * * * *
7 * * * * * * * *
8 * * * * * * * *
```

轮到 黑棋-X 落子

本次落子得分: 262

```
['F2', 'F3', 'F4', 'F5', 'F6']
```

F4

棋局分布情况:

```
R\C A B C D E F G H
1 * * * * * * * *
2 * * * * * * * *
3 * * * X O * * *
4 * * * X X X * *
5 * * * X O * * *
6 * * * * * * * *
7 * * * * * * * *
8 * * * * * * * *
```

请'白棋-O'输入: C3

棋局分布情况:

```
R\C A B C D E F G H
1 * * * * * * * *
2 * * * * * * * *
3 * * O O O * * *
4 * * * O X X * *
5 * * * X O * * *
6 * * * * * * * *
7 * * * * * * * *
8 * * * * * * * *
```

轮到 黑棋-X 落子

本次落子得分: 299

```
['C2', 'D2', 'C4', 'E2', 'F5', 'E6', 'D6']
```

D2

棋局分布情况:

```
R\C A B C D E F G H
1 * * * * * * * *
2 * * * X * * * *
3 * * O X X * * *
4 * * * X X X * *
5 * * * X O * * *
6 * * * * * * * *
7 * * * * * * * *
8 * * * * * * * *
```

请'白棋-O'输入: E2

棋局分布情况:

```
R\C A B C D E F G H
1 * * * * * * * *
2 * * * X O * * *
3 * * O X O * * *
4 * * * X O X * *
5 * * * X O * * *
6 * * * * * * * *
7 * * * * * * * *
8 * * * * * * * *
```

轮到 黑棋-X 落子

本次落子得分: 374

```
['F1', 'F2', 'F3', 'B4', 'B3', 'B2', 'F5', 'F6', 'D6']
```

F3

棋局分布情况:

```
R\C A B C D E F G H
1 * * * * * * * *
2 * * * X O * * *
3 * * O X X X * *
4 * * * X X X * *
5 * * * X O * * *
6 * * * * * * * *
7 * * * * * * * *
```

```
请'白棋-o'输入: c5
棋局分布情况:
R\c A B C D E F G H
1 * * * * * * * *
2 * * * X O * * *
3 * * O X X X * *
4 * * * X X X * *
5 * * O O * * *
6 * * * * * * *
7 * * * * * * *
8 * * * * * * *
轮到 黑棋-X 落子
本次落子得分: 374
['E1', 'F1', 'F2', 'D1', 'B4', 'B3', 'B2', 'D6', 'C6', 'B6', 'E6', 'F6']
D1
棋局分布情况:
R\c A B C D E F G H
1 * * * X * * * *
2 * * * X X * * *
3 * * O X X X * *
4 * * * X X X * *
5 * * O O * * *
6 * * * * * * *
7 * * * * * * *
8 * * * * * * *
```

2 评测指标展示及分析（机器学习实验必须有此项，其它可分析运行时间等）

|-----如有优化，请重复1, 2, 分析优化后的算法结果-----|

- 由于本人下棋水平有限，优化时选择了一款单机黑白棋游戏作为有一定下棋水平的对手来训练AI，我充当中介分别在单机游戏和我的程序间输入落子位置
- 优化过程：初始解无法击败单机游戏（结果为白棋胜），优化后可以击败单机游戏（结果为黑棋胜）

中午12:04

蓝牙 VPN HD HD WiFi 6

[《隐私政策》](#) 和 [《用户协议》](#)

全新版

黑白棋



单机
模式

联机
模式

再按一次取消按键屏蔽



设置



反馈

游戏。注意自我保护，谨防受骗上当。适度游戏益脑，沉迷

≡

□

<

中午12:04

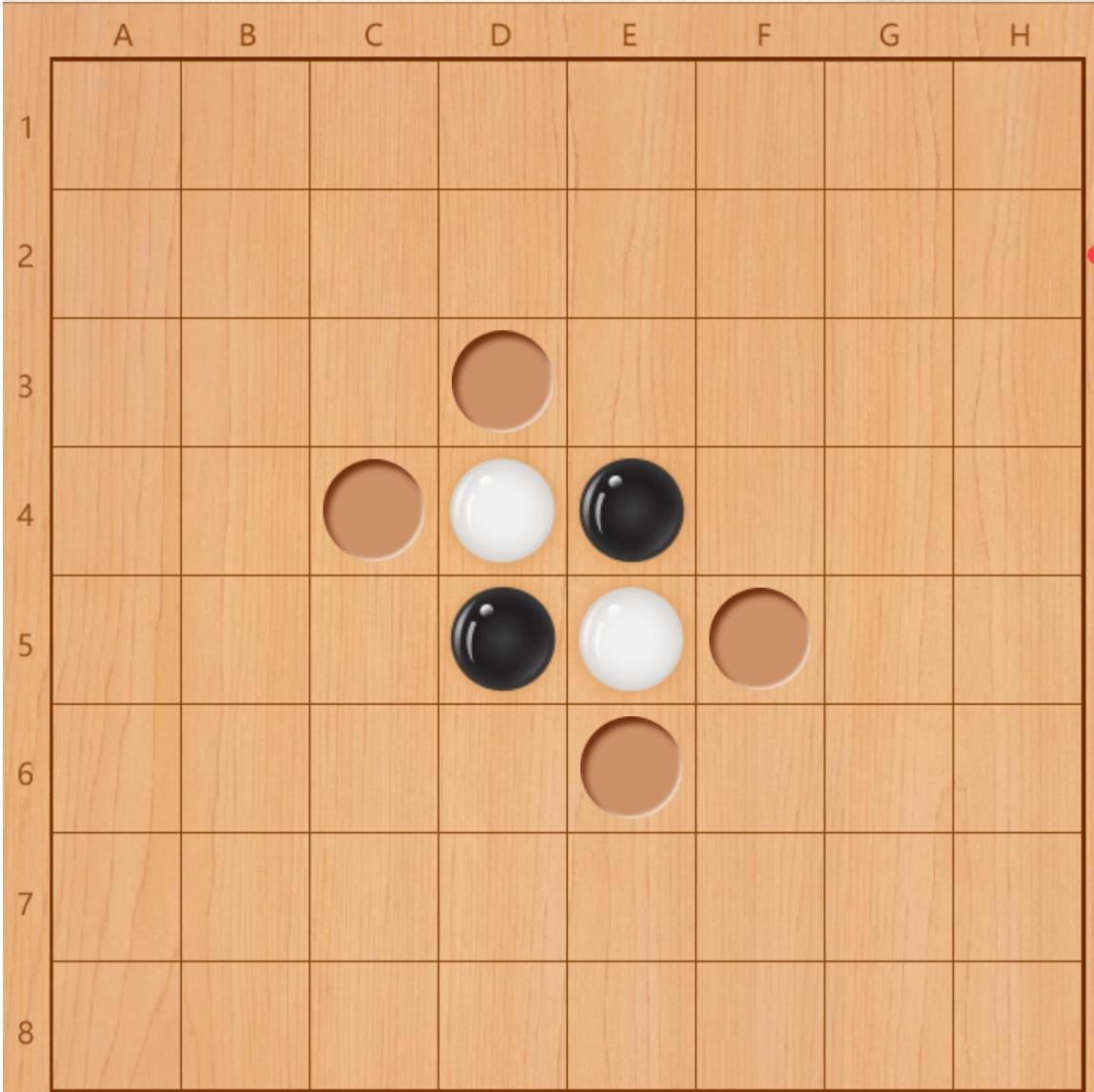
蓝牙 VPN HD HD 6G

玩家

电脑

2 : 2

普通



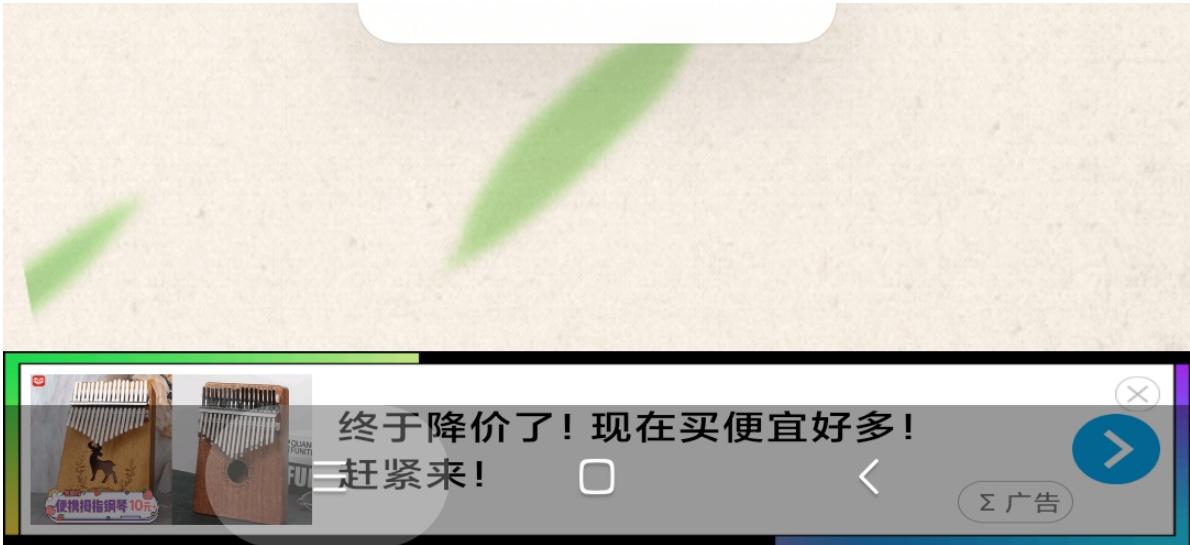
离开

悔棋

认输

重玩

再按一次取消按键屏蔽



初始解的情况

```
x[0] = 0.035940 #gapweight  
x[1] = 3.805478 #cornerweight  
x[2] = 48.281880 #mobilityWeight
```

```
---End of the game  
  
棋局分布情况:  
R\C A B C D E F G H  
1 X X 0 0 0 0 0 0 0  
2 0 0 0 0 0 0 0 0 0  
3 0 0 0 0 0 0 0 X  
4 0 0 0 0 0 0 0 X  
5 0 0 X 0 0 0 0 X  
6 X 0 X 0 0 0 0 X  
7 X X 0 X X 0 0 X  
8 0 0 0 0 0 0 0 X  
  
胜者为白棋  
PS C:\Users\Administrator> 
```

无法击败单机游戏

更改模拟退火算法参数，导出若干组解

```
tInitial = 100.0  
tFinal   = 70  
alpha     = 0.8          # 降温参数  
innerLoopLen = 5         # 内循环运行次数  
step      = 0.5          # 搜索步长
```

模拟退火算法参数设置

```
优化结果:  
x[0] = 0.015827  
x[1] = 2.819708  
x[2] = 40.831352
```

...

成功优化的结果

```
tInitial = 100.0  
tFinal   = 60  
alpha     = 0.8          # 降温参数  
innerLoopLen = 5         # 内循环运行次数  
step      = 0.5          # 搜索步长
```

模拟退火算法参数设置

```
优化结果：  
    x[0] = 0.014305  
    x[1] = 2.240337  
    x[2] = 45.051150  
PS C:\Users\Administrator> 
```

由模拟退火算法得到的新解

```
---End of the game  
  
棋局分布情况：  
R\C A B C D E F G H  
1   X O X X X X X X  
2   X O O X O O O X  
3   X O O O O X O X  
4   X O O O O X O X  
5   X O O X O X O X  
6   X O O X O X O X  
7   X X X O X O O O  
8   X X X X X X O O  
胜者为黑棋  
PS C:\Users\Administrator> 
```

成功击败单机游戏，AI得到优化

四、参考资料

[https://blog.csdn.net/weixin_42165981/article/details/103263211?
ops_request_misc=%257B%2522request%255Fid%2522%253A%252216493146771678026984415%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=16493146771678026984415&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-](https://blog.csdn.net/weixin_42165981/article/details/103263211?ops_request_misc=%257B%2522request%255Fid%2522%253A%252216493146771678026984415%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=16493146771678026984415&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-)

[2~all~top_positive~default-1-103263211.142^v6^control,157^v4^control&utm_term=alpha-beta剪枝算法&spm=1018.2226.3001.4187](#)

<https://blog.csdn.net/u012074597/article/details/80105676>

https://blog.csdn.net/coffee_cream/article/details/51754484