**程序报告**

学号： 2113997 姓名：齐明杰

1. **问题重述**

（简单描述对问题的理解，从问题中抓住主干，必填）

====================================================================

黑白棋 (Reversi)，也叫苹果棋，翻转棋，是一个经典的策略性游戏。

一般棋子双面为黑白两色，故称“黑白棋”。因为行棋之时将对方棋子翻转，则变为己方棋子，故又称“翻转棋” (Reversi) 。

棋子双面为红、绿色的称为“苹果棋”。它使用 8x8 的棋盘，由两人执黑子和白子轮流下棋，最后子多方为胜方。

实验要求:

1、使用 『蒙特卡洛树搜索算法』 实现 miniAlphaGo for Reversi。

2、使用 Python 语言。

3、算法部分需要自己实现，不要使用现成的包、工具或者接口。

1. **设计思想**

（所采用的方法，有无对方法加以改进，该方法有哪些优化方向（参数调整，框架调整，或者指出方法的局限性和常见问题），伪代码，理论结果验证等… **思考题，非必填**）

====================================================================

**方法**：本代码采用了蒙特卡洛树搜索（Monte Carlo Tree Search，MCTS）算法，结合Roxanne策略进行黑白棋游戏的AI决策。蒙特卡洛树搜索是一种自适应的搜索方法，可以在有限时间内找到近似最优解。Roxanne策略则是通过对落子优先级进行排序，提高了搜索过程中的效率。

**改进**：在模拟扩展阶段，代码将随机策略替换为Roxanne策略，使得搜索过程更加高效。

**伪代码**：

1. 初始化蒙特卡洛树节点和AI玩家

2. 在时间限制内进行蒙特卡洛树搜索：

1) 选择最有潜力的子节点

2) 扩展选择的节点

3) 采用Roxanne策略进行模拟

4) 反向传播更新节点信息

3. 根据搜索结果返回最佳落子位置

**方法局限性**：MCTS受时间限制影响较大，因此可能无法找到全局最优解。此外，在某些情况下，Roxanne策略可能限制了搜索空间，导致未能找到最佳解。

**优化方向**：

参数调整：尝试调整权重表以改进搜索效果；

框架调整：可以尝试引入其他启发式搜索算法，例如alpha-beta剪枝等。

1. **代码内容**

（能体现解题思路的主要代码，有多个文件或模块可用多个"===="隔开，必填）

**main.py**

====================================================================

import random

from math import log, sqrt

from time import time

from copy import deepcopy

from game import Game

from board import Board

class SilentGame(Game):

def \_\_init\_\_(self, black\_player, white\_player, board = Board(), current\_player = None):

super().\_\_init\_\_(black\_player, white\_player)

self.board = deepcopy(board)

self.current\_player = current\_player

def run(self):

winner = None

diff = -1

while True:

self.current\_player = self.switch\_player(self.black\_player, self.white\_player)

color = "X" if self.current\_player == self.black\_player else "O"

legal\_actions = list(self.board.get\_legal\_actions(color))

if len(legal\_actions) == 0:

if self.game\_over():

winner, diff = self.board.get\_winner()

break

else:

continue

action = self.current\_player.get\_move(self.board)

if action is None:

continue

else:

self.board.\_move(action, color)

if self.game\_over():

winner, diff = self.board.get\_winner()

break

return winner, diff

class RoxannePlayer(object):

def \_\_init\_\_(self, color):

"""

Roxanne策略初始化

:param roxanne\_table: 从上到下依次按落子优先级排序

:param color: 执棋方

"""

self.roxanne\_table = [

['A1', 'H1', 'A8', 'H8'],

['C3', 'F3', 'C6', 'F6'],

['C4', 'F4', 'C5', 'F5', 'D3', 'E3', 'D6', 'E6'],

['A3', 'H3', 'A6', 'H6', 'C1', 'F1', 'C8', 'F8'],

['A4', 'H4', 'A5', 'H5', 'D1', 'E1', 'D8', 'E8'],

['B3', 'G3', 'B6', 'G6', 'C2', 'F2', 'C7', 'F7'],

['B4', 'G4', 'B5', 'G5', 'D2', 'E2', 'D7', 'E7'],

['B2', 'G2', 'B7', 'G7'],

['A2', 'H2', 'A7', 'H7', 'B1', 'G1', 'B8', 'G8']

]

self.color = color

def roxanne\_select(self, board):

"""

采用Roxanne 策略选择落子策略

:return: 落子策略

"""

action\_list = list(board.get\_legal\_actions(self.color))

if len(action\_list) == 0:

return None

else:

for move\_list in self.roxanne\_table:

random.shuffle(move\_list)

for move in move\_list:

if move in action\_list:

return move

def get\_move(self, board):

"""

采用Roxanne 策略进行搜索

:return: 落子

"""

action = self.roxanne\_select(board)

return action

class TreeNode():

"""

蒙特卡洛树节点

"""

def \_\_init\_\_(self, parent, color):

self.parent = parent

self.w = 0

self.n = 0

self.color = color

self.child = dict()

class AIPlayer:

"""

AI 玩家

"""

def \_\_init\_\_(self, color, time\_limit = 10, c\_param = sqrt(2)):

"""

玩家初始化

:param color: 下棋方，'X' - 黑棋，'O' - 白棋

"""

self.c\_param = c\_param

self.time\_limit = time\_limit

self.tick = 0

self.sim\_black = RoxannePlayer('X')

self.sim\_white = RoxannePlayer('O')

self.color = color

def mcts(self, board):

"""

蒙特卡洛树搜索，在时间限制范围内，拓展节点搜索结果

:return: 选择最佳拓展

"""

root = TreeNode(None, self.color)

# 设定一个时间停止计算，限定规模

while time() - self.tick < self.time\_limit:

sim\_board = deepcopy(board)

choice = self.select(root, sim\_board)

self.expand(choice, sim\_board)

winner, diff = self.simulate(choice, sim\_board)

back\_score = [1, 0, 0.5][winner]

if choice.color == 'X':

back\_score = 1 - back\_score

self.back\_prop(choice, back\_score)

best\_n = -1

best\_move = None

for k in root.child.keys():

if root.child[k].n > best\_n:

best\_n = root.child[k].n

best\_move = k

return best\_move

def select(self, node, board):

"""

蒙特卡洛树搜索，节点选择

:return: 搜索树向下递归选择子节点

"""

if len(node.child) == 0:

return node

else:

best\_score = -1

best\_move = None

for k in node.child.keys():

if node.child[k].n == 0:

best\_move = k

break

else:

N = node.n

n = node.child[k].n

w = node.child[k].w

# 随着访问次数的增加，加号后面的值越来越小，因此我们的选择会更加倾向于选择那些还没怎么被统计过的节点

score = w / n + self.c\_param \* sqrt(log(N) / n)

if score > best\_score:

best\_score = score

best\_move = k

board.\_move(best\_move, node.color)

return self.select(node.child[best\_move], board)

def expand(self, node, board):

"""

蒙特卡洛树搜索，节点扩展

"""

op\_color = 'O' if node.color == 'X' else 'X'

for move in board.get\_legal\_actions(node.color):

node.child[move] = TreeNode(node, op\_color)

def simulate(self, node, board):

"""

蒙特卡洛树搜索，采用Roxanne策略代替随机策略搜索，模拟扩展搜索树

"""

if node.color == 'O':

current\_player = self.sim\_black

else:

current\_player = self.sim\_white

sim\_game = SilentGame(self.sim\_black, self.sim\_white, board, current\_player)

return sim\_game.run()

def back\_prop(self, node, score):

"""

蒙特卡洛树搜索，反向传播，回溯更新模拟路径中的节点奖励

"""

node.n += 1

node.w += score

if node.parent is not None:

self.back\_prop(node.parent, 1 - score)

def get\_move(self, board):

"""

根据当前棋盘状态获取最佳落子位置

:param board: 棋盘

:return: action 最佳落子位置, e.g. 'A1'

"""

self.tick = time()

if self.color == 'X':

player\_name = '黑棋'

else:

player\_name = '白棋'

print("请等一会，对方 {}-{} 正在思考中...".format(player\_name, self.color))

# -----------------请实现你的算法代码--------------------------------------

action = self.mcts(deepcopy(board))

# ------------------------------------------------------------------------

return action

1. **实验结果**

（实验结果，必填）

====================================================================





1. **总结**

（自评分析（是否达到目标预期，可能改进的方向，实现过程中遇到的困难，从哪些方面可以提升性能，模型的超参数和框架搜索是否合理等），**思考题，非必填**）

====================================================================

**是否达到目标预期**：本代码通过结合Roxanne策略和蒙特卡洛树搜索（MCTS）算法，实现了一个相对高效且智能的黑白棋AI。然而，对于棋力较高的对手来说，该AI可能无法展现出十分出色的表现。

**优化方向**：

1. 针对Roxanne策略，可以尝试通过机器学习方法对优先级表进行优化，提高落子位置选择的准确性。
2. 在蒙特卡洛树搜索过程中，可以尝试使用更复杂的评估函数或策略，例如基于深度学习的价值网络或策略网络，提高搜索效果。
3. 考虑使用并行计算或GPU加速来加快搜索速度，提高搜索的深度和广度。

**实现过程中遇到的困难**：

在实现蒙特卡洛树搜索时，可能遇到搜索速度较慢的问题，导致搜索深度受限，从而影响AI的棋力。

**模型的超参数和框架搜索是否合理**：

当前的超参数设置和框架搜索是基于经验的，可能并非最优。可以通过网格搜索、随机搜索或贝叶斯优化等方法，尝试寻找更优的超参数组合。此外，框架搜索过程可以尝试引入启发式算法（如遗传算法等）来寻找更优的搜索路径。