CS4386 Assignment 1 (Semester B, 2022-2023)

Name: JIANG XIN
Student ID: 56643608

Report:

My model is the monte carol search tree with a specific UCT.
Because the game is second hand must win, so it is better to consider different strategies based on the prior experience. (Prove in appendix)

After trying in min_max tree/negamax tree with pruning with a depth=10, it shows that although I'm the second hand, the tree will select the strategies must win with less score, and even my first can win in specific situation, it also will select the strategies lose less.

So consider as a score-based game, it is better to consider use Monte Carol Search Tree as the AI model. To win more score than opponents, it is better to use my score minus opponent's score as the evaluation score and with a reward for high evaluation score to encourage the AI try the high score strategy, which also can reduce the random factor causing high score.

Based on such an UCT method, it can win more with not smart AI and loss less with smart AI, and with enough simulation times, it will behavior like min-max tree when the depth is high.

```
#uct should based on score to scale the score gap with opponent large
o.uct = o.score/12+o.wins/(o.visit+0.001) +(2)*np.sqrt(np.log(self.visit+1)/(o.visit+0.001))
```

According to the , give a limitation on simulation which can save memory and time, even improve the simulation results. That's because for time limited, it is not need to try different action for high depth, less time trying even make more error.
Also, as I have considered use pruning to help the tree run more times and inherited the prior experience. But as trying, the time and memory cost of pruning is large which the effect even worse than using the time to simulate more for current situation.

```
while(time.time()-t<9.8):
    level+=1
    #depth limitation
    if level>6:
        s = self.quickout(board,player)
        score += s
        score = -score
        curr.backpropagation(score>0,score)
        break
```

Workflow:

1. Establish a Monte Carol Tree with depth = 6

```
#depth limitation
if level>6:
    s = self.quickout(board,player)
    score += s
    score = -score
    curr.backpropagation(score>0,score)
    break
#expansion
if curr.is_leaf():
    cells = self.eval.available_cells(board,player)
    curr.expansion(cells)
#backpropagation
if curr.sons == {}:
    if self.eval.full(board):
        score=-score
        if time.time()-t<9.8:
            curr.backpropagation(score>0,score)
        break
```

2. Simulation of playing chess in random

```
def quickout(self,board,player):
    #random play chess to evaluate the result
    score = 0
    oppoent = "O" if player=="X" else "X"
    cells,cells_oppoent = self.eval.available_cells_both(board,player)
    l_m,l_o = len(cells),len(cells_oppoent)
    seq_me = self.random_arr(l_m)
    seq_oppoent = self.random_arr(l_o)


    i = 0
    while i < l_m and i < l_o: ...
    if i < l_m: ...
    if i < l_o: ...
    return score
```

3. Select the highest score son node.

```
#highest score son
for k,e in self.root.sons.items():
    s = e.score
    if s>=m_s:
        m_s = s
        m_k = k
return m_k
```

4. Make the movement.

```
start_time = time.time()
next_move = Tree_56643608().stategy(state,player,t=start_time)
return next_move
```

Appendix:

(Prove: Consider each board situation with N chess occupied, N~(0,36), the combination of x chess can be calculate by $P(N=x) = {}_{18}C_{x//2} * {}_{18}C_{x//2}$ if x == even else ${}_{18}C_{x//2} * {}_{18}C_{x//2-1}$, All the combinations is $\sum P(X=N) \approx \sum {}_{18}C_{x//2}{}^{2} \approx {}_{36}C_{18}$, build the Zobrist hash table from X=36 to X=0, X = n can be calculated by X = n+1, so actually all the board result can be totally calculated. After do so, I found the score of X=0 is negative)