**Jinqi Cheng Assignment 4 Part 1**

**Problem 1**

Answer:

2 : 0.0625

3 : 0.125

4 : 0.1875

5 : 0.25

6 : 0.1875

7 : 0.125

8 : 0.0625

Work:

count = 0 #total number of solutions

dic = dict() #key: 2 to 8, value: number of key appearances

#dice 1

for i in range(1,5):

#dice2

for j in range(1,5):

s = i+j

if s in dic:

dic[s] += 1

else:

dic[s] = 1

count+=1

for key,value in dic.items():

print(key,": ",value/count)

**Problem 2**

$\begin{aligned} utility(Γ) &= P(A) \times 10 + P(B) \times 15 + P(C) \times 29 \ &= 0.25 \times 10 + 0.50 \times 15 + 0.25 \times 29 \ &= 17.25 \end{aligned}$ $\begin{aligned} utility(Ψ) &= P(D) \times 14 + P(E) \times 18 \ &= 0.45 \times 14 + 0.55 \times 18\ &= 16.2 \end{aligned}$ $\because utility(Γ) > utility(Ψ)$ $\therefore$ the player should choose $Γ$

**Problem 3**

We can create a function called isStalemate(). This function returns a boolean value and take two parameters: self - the checkerboard dic - a dictionary that stores number of times that each board state appears dic - key: checkerboard repr, value: number of appearances This function should be called after each move() is performed The following is a piece of pseudocode.

def isStalemate(self, dic):

#currentboard is a string

currboard = self.\_\_repr\_\_()

if currboard in dic:

dic[currboard] += 1

else:

dic[currboard] = 1

if dic[currboard] >= 3:

return true

return false

We can optimize this solution by clear the dic after a piece been captured, since any board state after that cannot be the same as the state before capturing the piece. This saves the memory consumption by avoiding the hashmap been too big. We can perform this inside move() method:

def move(self, move, validate=[], verbose=False):

...

if a piece is captured:

dic.clear()

...

**Hsuan Yu, Liu RedID: 823327369**

**Part 1:**

**1.**

Ans:

2: (1+1) 🡺 1/4\*1/4 = 1/16

3: (1+2) \*2 🡺 (1/4\*1/4) \*2 = 1/8

4: (1+3) \*2, (2+2) 🡺(1/16) \*2 + (1/16) = 3/16

5: (1+4) \*2, (2+3) \*2 🡺(1/16) \*4 = 4/16 = 1/4

6: (2+4) \*2, (3+3) 🡺 (1/16) \*2 + (1/16) \*1 = 3/16

7: (3+4) \*2 🡺 (1/16) \*2 = 2/16

8: (4+4) 🡺 1/4\*1/4 = 1/16

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Values | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| Probability | 1/16 | 1/8 | 3/16 | 1/4 | 3/16 | 1/8 | 1/16 |

**2.**

Ans:

Γ Expectation = 10\*0.25+15\*0.5+29\*0.25 = 2.5+ 7.5 +7.25 =17.25

Ψ Expectation = 14\*0.45 + 18\*0.55=6.3 + 9.9 = 16.2

To maximize utility, a player will select Γ.

**3.**

Ans:

Use a dictionary to record configurations. Key is “checkboard class” and value is the times of a configuration. For example, configHistory = [checkerboard01: 2] 🡺 checkerboard01 appears 2 times.

After every move, the program checks whether the current configuration exists in configHistory. If it exists, add one to its value. If it doesn’t exist, then add a new key which is the current configuration, and the value is one into the configHistory. If a configuration shows 3 times, then the configuration is stalemated.

**ai.py**

import abstractstrategy

import checkerboard

class Strategy(abstractstrategy.Strategy):

    ## Alpha-beta pruning minimax search.

    def \_\_init\_\_(self, player, game, maxplies):

        super().\_\_init\_\_(player,game, maxplies)

        # super.\_\_init\_\_(player=player,game=game, maxplies=maxplies)

    def utility(self, board):

        "Return the utility of the specified board"

        playerId = board.playeridx(self.maxplayer)

        otherId = board.playeridx(self.minplayer)

        return board.pawnsN[playerId] + board.kingsN[playerId]\*4 - board.pawnsN[otherId] - board.kingsN[otherId]\*4

        # raise NotImplementedError("Subclass must implement")

    def play(self, board):

        """"play - Make a move

        Given a board, return (newboard, action) where newboard is

        the result of having applied action to board and action is

        determined via a game tree search (e.g. minimax with alpha-beta

        pruning).

        """

        print(self.maxplayer,' thinking using Alpha-Beta pruning strategy...')

        action =  self.maxValue(board,0, float("-inf"), float("inf"))[1]

        if not action:

            return (board, action)

        else:

            return (board.move(action), action)

    def maxValue(self, board,  depth, alpha, beta):

        if depth >= self.maxplies:

            return (self.utility(board),None)

        else:

            actions = board.get\_actions(self.maxplayer)

            val = float('-inf')

            move = None

            for action in actions:

                newBoard = board.move(action)

                val = max(val, self.minValue(newBoard, depth+1, alpha, beta)[0])

                if val >= alpha:

                    move = action

                if val >= beta:

                    break

                else:

                    alpha = max(alpha, val)

            return (val, move)

    def minValue(self, board,  depth, alpha, beta):

        if depth >= self.maxplies:

            return (self.utility(board),None)

        else:

            actions = board.get\_actions(self.minplayer)

            val = float('inf')

            move = None

            for action in actions:

                newBoard = board.move(action)

                val = min(val, self.maxValue(newBoard, depth+1, alpha, beta)[0])

                if val >= beta:

                    move = action

                if val <= alpha:

                    break

                else:

                    beta = min(beta, val)

            return (val, move)

**checkers.py**

'''

@author: mroch

'''

# Game representation and mechanics

import checkerboard

import imp

import sys

major = sys.version\_info[0]

minor = sys.version\_info[1]

modpath = "\_\_pycache\_\_/tonto.cpython-{}{}.pyc".format(major, minor)

tonto = imp.load\_compiled("tonto", modpath)

# human - human player, prompts for input

import human

import boardlibrary # might be useful for debugging

from timer import Timer

def Game(red=human.Strategy, black=tonto.Strategy,

         maxplies=10, init=None, verbose=True, firstmove=0):

    """Game(red, black, maxplies, init, verbose, turn)

    Start a game of checkers

    red,black - Strategy classes (not instances)

    maxplies - # of turns to explore (default 10)

    init - Start with given board (default None uses a brand new game)

    verbose - Show messages (default True)

    firstmove - Player N starts 0 (red) or 1 (black).  Default 0.

    """

    # Don't forget to create instances of your strategy,

    # e.g. black('b', checkerboard.CheckerBoard, maxplies)

    ##----------------------------------------- Implement

    gameboard = checkerboard.CheckerBoard()

    redPlayer = red('r', checkerboard.CheckerBoard, maxplies)

    blackPlayer = black('b', checkerboard.CheckerBoard, maxplies)

    turn = True if firstmove == 0 else False

    if verbose:

        clock = Timer()

    move\_cnt= 0

    while not gameboard.is\_terminal()[0]:

        move\_cnt+=1

        if turn:

            if verbose:

                gameboard = verboseMode(redPlayer,gameboard,clock)

            else:

                gameboard = ClearMode(redPlayer,gameboard)

            turn = not turn

        else:

            if verbose:

                gameboard = verboseMode(blackPlayer,gameboard,clock)

            else:

                gameboard = ClearMode(blackPlayer,gameboard)

            turn = not turn

    if verbose:

        print('Final board')

        print(gameboard)

        winner = gameboard.is\_terminal()

        if not winner[1]:

            print('Game is a draw')

        else:

            print('Winner is', winner[1])

    return gameboard

    # raise NotImplemented

def verboseMode(player, gameboard, clock):

    print('Player {} turn'.format(player.maxplayer))

    print(gameboard)

    move\_clock = Timer()

    (\_,move) = player.play(gameboard)

    elapsed\_move = move\_clock.elapsed\_s()

    elapsed\_all = clock.elapsed\_min()

    if not move:

        print('Move \'None\'','by {}:'.format(player.maxplayer),' None ', 'Result:')

    else:

        print('Move ', '{:-5}'.format(gameboard.movecount),'by {}:'.format(player.maxplayer),gameboard.get\_action\_str(move), 'Result:')

        gameboard = gameboard.move(move)

    print(gameboard)

    print('Pawn/King count: r {} R {} b {} B {}'.format(gameboard.pawnsN[0], gameboard.kingsN[0], gameboard.pawnsN[1], gameboard.kingsN[1]))

    print('Move: {:.3} s, Game: {:.4} min\n'.format(elapsed\_move, elapsed\_all))

    return gameboard

def ClearMode(palyer, gameboard):

    (\_,move) = palyer.play(gameboard)

    if move:

        gameboard = gameboard.move(move)

    return gameboard

if \_\_name\_\_ == "\_\_main\_\_":

    # Game(init=boardlibrary.boards["multihop"])

    #Game(init=boardlibrary.boards["StrategyTest1"])

    #Game(init=boardlibrary.boards["EndGame1"], firstmove = 1)

    Game()

**Pair Programming Equitable Participation & Honesty Affidavit**

We the undersigned promise that we have in good faith attempted to follow the principles of pair programming. Although we were free to discuss ideas with others, the implementation is our own. We have shared a common workspace and taken turns at the keyboard for the majority of the work that we are submitting. Furthermore, any non programming portions of the assignment were done independently. We recognize that should this not be the case, we will be subject to penalties as outlined in the course syllabus.

Jinqi Cheng 03/22/2020

Pair Programmer 1 (sign your name, then date it)   
  
  
  
  
Hsuan Yu Liu 03/22/2020

Pair Programmer 2 (sign your name, then date it)

**Pair Programming Equitable Participation & Honesty Affidavit**

We the undersigned promise that we have in good faith attempted to follow the principles of pair programming. Although we were free to discuss ideas with others, the implementation is our own. We have shared a common workspace and taken turns at the keyboard for the majority of the work that we are submitting. Furthermore, any non programming portions of the assignment were done independently. We recognize that should this not be the case, we will be subject to penalties as outlined in the course syllabus.

Jinqi Cheng 03/22/2020

Pair Programmer 1 (sign your name, then date it)   
  
  
  
Hsuan Yu Liu 03/22/2020

Pair Programmer 2 (sign your name, then date it)