Resultados

A*

Ao implementar o algoritmo A* com a heurística dada, os resultados obtidos em diversos testes revelavam um algoritmo que só se importava em maximizar os pontos. Por essa razão, nem sempre realizava a jogada mais adequada pois, por exemplo, em vez de impedir a vitória do adversário, e consequentemente a sua derrota, dava preferência em jogar numa coluna onde iria obter mais pontuação.

Posto isto, implementamos uma segunda heurística que considera se o adversário irá ganhar. Assim, o algoritmo dará preferência à jogada que evita a derrota, em vez de só querer somar mais pontos. Entre ganhar e não perder, vai escolher a primeira opção. Esta heurística melhora o algoritmo pois para podermos ganhar posteriormente temos primeiro de evitar que o nosso oponente ganhe já de seguida.

Em qualquer jogo A* vs A* o resultado vai ser sempre o mesmo, porque não há variação nenhuma nas iterações. O ‘X’ é o primeiro a jogar e quem ganha, em 30 jogadas, é o ‘O’ .

MCTS

Iniciamos os testes do MCTS com um limite de 5 segundos de pesquisa e sem limite de iterações. No estado inicial do jogo, nos 5 segundos eram realizadas 17.000 iterações e as decisões tomadas pelo algoritmo correspondiam com as que um humano também teria. Ao longo do jogo e à medida que o tabuleiro ia sendo preenchido, nos 5 segundos, entre 30.000 e 50.000 iterações eram realizadas. Mas, as decisões que eram tomadas já não correspondiam ao que seria esperado.

Então, posto isto, implementamos um número limite de iterações realizadas possíveis. Com isto, no início com o tabuleiro vazio, tínhamos 5 segundos de limite e X iterações, em menos de 5 segundos, para um tabuleiro cada vez com mais peças colocadas.

Após vários testes com diversos limites, chegamos à conclusão que com 22.500 iterações o algoritmo tomava decisões coerentes com o que era esperado. Algumas vezes, o algoritmo não vence na primeira oportunidade que tem durante o jogo e esse problema não conseguimos solucionar.

Chegamos a um equilíbrio entre começar em primeiro (‘X’) ou começar em segundo (‘O’), pois o número de vitórias, em 50 testes realizados, foram 25 para cada um sem que tenha ocorrido qualquer empate e com uma média de 34 rondas por teste.

Código em cada ficheiro

Game4InLine.py

Código com a lógica do jogo

Contém a classe "Game4InLine" onde está implementado o jogo e também o algoritmo A*

```
from copy import deepcopy

#Human start and uses 'X' (player 1)
```

```
#AI go second and uses 'O' (player 2)

class Game4InLine:
        def ____init_____ (self,row,col):
                '''
                initialize the game, board and all other components to be able
to play
                '''
                self.rows = row
                self.cols = col
                self.board = [['-' for _ in range(col)] for _ in range(row)]
#matrix representing the board, initialized full with '-'
                self.placed = [0 for _ in range(col)] #store the num of pieces per column, initialized with
0s
                self.pieces = ['X','O'] # different pieces
                self.turn = 0 # to know next player, switch between 0 and 1
                self.round = 0


        def legal_moves(self):
                '''
                returns a list with the columns that are possible to play, this is, that are not full
                '''
                legal=[]

                for i in range(self.cols):
                        if self.placed[i]<self.rows: #placed[i]<rows means that the number of pieced
placed in the row-i is less than the max pieced that are possible to place
                                legal.append(i) return

                legal


        def childs(self):
                '''
                this funcion returns a list for the possible childs based on the current state of the
board
                the list is made of lists with the format -> [child: Game4InLine, col: int]
                where child is the new game made from the current and played at the col selected
                '''
                moves=self.legal_moves() # returns the possible moves
                children = []

                for col in moves:
                        temp=deepcopy(self)
                        children.append([temp.play(col),col])
```

```
            return children


      def play(self,col:int): #funcion to place pieces based on turn and column
            '''
            it is guaranteed that the col given is not full

            given a col it will place the piece, X or O based on turn the piece will be
            placed at the bottom of the column
            and updated all the data regarding turn, round and placed[] '''
            self.board[self.rows-self.placed[col]-1] [col]=self.pieces[self.turn]
            self.placed[col]+=1 self.round+=1
            self.turn = 1 if self.turn%2==0 else 0 #change turn
            return self


      def isFinished(self,col):
            '''
            return 2 if game is a draw, True if last move was a winning one , False to keep
playing
            (we return True for win and 2 for draw because 2!=True but "if isFinished()" will be
considered true if the return is 2 and we use it for diferenciate from draw or win)

            based on the game and the column played last we analyze if there is a sequence of
4(or more) of the same type of piece placed
            we start at the position of the last played piece and check vertical, horizontal and
both diagonals
            if there is no sequence of 4(or more) but the board is full (when round == rows*cols)
we return 2 for a draw
            '''
            played = self.pieces[self.turn-1]
            row = self.rows - max(self.placed[col],1)

            #Check       Vertical        |
            consecutive = 1  # | tmprow
            = row     # |
            while tmprow+1 < self.rows and self.board[tmprow+1][col] ==
played:
                  consecutive+=1
                  tmprow+=1
            if consecutive >= 4: return
                  True
```

```
        # Check horizontal   ----
        consecutive = 1 tmpcol =
        col
        while tmpcol+1 < self.cols and self.board[row][tmpcol+1] ==
played:
            consecutive += 1
            tmpcol += 1 tmpcol =
        col
        while tmpcol-1 >= 0 and self.board[row][tmpcol-1] == played: consecutive += 1
            tmpcol -= 1

        if consecutive >= 4: return
            True


        # Check diagonal      1          \
        consecutive = 1                 #  \
        tmprow = row                     #  \
        tmpcol = col                      #  \
        while tmprow+1 < self.rows and tmpcol+1 < self.cols and
self.board[tmprow+1][tmpcol+1] == played:
            consecutive += 1
            tmprow += 1
            tmpcol += 1
        tmprow = row tmpcol
        = col
        while tmprow-1 >= 0 and tmpcol-1 >= 0 and self.board[tmprow-1] [tmpcol-1] ==
played:
            consecutive += 1
            tmprow -= 1
            tmpcol -= 1

        if consecutive >= 4: return True


        # Check diagonal      2          /
        consecutive = 1               #  /
        tmprow = row                 #  /
        tmpcol = col                #  /
        while tmprow-1 >= 0 and tmpcol+1 < self.cols and
self.board[tmprow-1][tmpcol+1] == played:
            consecutive += 1
            tmprow -= 1
            tmpcol += 1
        tmprow = row tmpcol
        = col
        while tmprow+1 < self.rows and tmpcol-1 >= 0 and
self.board[tmprow+1][tmpcol-1] == played:
```

```
                consecutive += 1
                tmprow += 1
                tmpcol -= 1

        if consecutive >= 4: return True


        # Check for draw
        if(self.round==(self.rows*self.cols)): #maybe change this to other function
                return 2

        return False


    def heuristic_extra(game,col):
        '''
        this heuristic was made to make A* more defensive
        where it will prioritize not losing rather than getting max point from the given heuristic
for the project

        it gives points based on col played and the player turn
        if it is a win move it will give -512 for O and 512 for X as the given heuristic
        but if it has a chance to defend from a lose, for example: O played and he made a
play that got a sequence of 3-X and 1-O
        it will consider that it was a good defence move because it made impossible for the X
to win next turn

        the point are given as follow:
        -500 for 3 Xs and 1 O and it is O turn
        -100 for 2 Xs and 1 O and it is O turn
        100 for 2 Os and 1 X and it is X turn
        500 for 3 Os and 1 X and it is X turn
        0 for else '''
        row = game.rows-game.placed[col]

        #caso for uma jogada para ganhar
        if game.isFinished(col):
                return -512 if game.turn%2==0 else 512


        #caso for uma jogada para nao perder
        points=0
        #horizontal
        for tpmcol in range(max(0,col-3),col+1): if
                tpmcol+3>game.cols-1: continue count_X=0
```

```
        count_O=0
        for j in range(4):
                if (game.board[row][tpmcol+j] == "X"): count_X+=1
                if (game.board[row][tpmcol+j] == "O"): count_O+=1

        #dar pontos
        h_value = getPoints_extra(game,count_X,count_O) if
        abs(h_value) == 500:
                return -500 if game.turn%2==0 else 500 elif h_value !=
        0:
                points=h_value



#vertical
count_X=0
count_O=0
for     i     in     range(0,min(4,game.rows-row)):     if
        (game.board[row+i][col] == "X"):
                count_X+=1
        if (game.board[row+i][col] == "O"): count_O+=1

#dar pontos
h_value = getPoints_extra(game,count_X,count_O) if
abs(h_value) == 500:
        return -500 if game.turn%2==0 else 500 elif h_value !=
0:
        points=h_value



#diagonal 1
tmpcol = col
tmprow = row
i=0
while(i<3 and tmprow>0 and tmpcol>0): tmpcol-=1
        tmprow-=1 i+=1
while i>=0 and tmprow+3<game.rows and tmpcol+3<game.cols: i-=1
        count_X=0
        count_O=0
        for h in range(4):
                if (game.board[tmprow+h][tmpcol+h] == "X"): count_X+=1
                if (game.board[tmprow+h][tmpcol+h] == "O"):
```

```
                    count_O+=1

            h_value = getPoints_extra(game,count_X,count_O) if
            abs(h_value) == 500:
                return -500 if game.turn%2==0 else 500 elif h_value !=
            0:
                points=h_value

            tmpcol+=1
            tmprow+=1


        #diagonal 2
        tmpcol = col
        tmprow = row
        i=0
        while i<3 and tmprow<game.rows-1 and tmpcol>0: tmpcol-=1
            tmprow+=1 i+=1

        while i>=0 and tmprow-3>=0 and tmpcol+3<game.cols: i-=1
            count_X=0
            count_O=0
            for h in range(4):
                if (game.board[tmprow-h][tmpcol+h] == "X"): count_X+=1
                if (game.board[tmprow-h][tmpcol+h] == "O"): count_O+=1

            h_value = getPoints_extra(game,count_X,count_O) if
            abs(h_value) == 500:
                return -500 if game.turn%2==0 else 500 elif h_value !=
            0:
                points=h_value

            tmpcol+=1
            tmprow-=1


        return abs(points)*(-1) if game.turn%2==0 else abs(points) def


    heuristic_points(game,col):
        '''
        this is the given heuristic for the project, it takes col as an input but it doesn't use
it, this happen because the other heuristic needs col
        and we use a lambda funcion on our A* that takes the sum of
```

```
both heuristisc

            the point are given as follow:
            -50 for three Os, no Xs,
            -10 for two Os, no Xs,
            - 1 for one O, no Xs,
            0 for no tokens, or mixed Xs and Os,
            1 for one X, no Os,
            10 for two Xs, no Os,
            50 for three Xs, no Os.

            and depending on whose turn is to play (+16 for X, -16 for O) '''

            points = 16 if game.pieces[game.turn] == 'X' else -16
            #horizontal
            for i in range(game.rows):
                  for j in range(game.cols-3): count_X=0
                        count_O=0
                        for h in range(j,j+4):
                              if (game.board[i][h] == "X"): count_X+=1
                              if (game.board[i][h] == "O"): count_O+=1

                        h_value = getPoints(count_X,count_O) if
                        abs(h_value) == 512:
                              return h_value else:
                              points+=h_value

            #vertical
            for i in range(game.cols):
                  for j in range(game.rows-3): count_X=0
                        count_O=0
                        for h in range(j,j+4):
                              if (game.board[h][i] == "X"): count_X+=1
                              if (game.board[h][i] == "O"): count_O+=1

                        h_value = getPoints(count_X,count_O) if
                        abs(h_value) == 512:
                              return h_value else:
                              points+=h_value

            #diagonal 1
```

```
        for i in range(game.rows-3):
            for j in range(game.cols-3): count_X=0
                count_O=0
                for h in range(4):
                    if (game.board[i+h][j+h] == "X"): count_X+=1
                    if (game.board[i+h][j+h] == "O"): count_O+=1

                h_value = getPoints(count_X,count_O) if
                abs(h_value) == 512:
                    return h_value else:
                    points+=h_value

        #diagonal 2
        for  i  in  range(game.rows-3): for j in
            range(3,game.cols):
                count_X=0 count_O=0
                for h in range(4):
                    if (game.board[i+h][j-h] == "X"): count_X+=1
                    if (game.board[i+h][j-h] == "O"): count_O+=1

                h_value = getPoints(count_X,count_O) if
                abs(h_value) == 512:
                    return h_value else:
                    points+=h_value

        return points


    def A_star(self,heuristic):
        '''
        As in this project our A* only looks for its next best play without going in depht for a
possible move from its the oponent we don't need to do a loop until the game is finished
        We will use a list to store [heuristic(child),col] and sort it so the best play for 'O' is first
and for 'X' is last
        A* will play as 'O' when vs human, so the lower the score the best (due to our
heuristic setup)
        it returns the col from best child based on the turn '''
        childs=self.childs()
        points_col=[]   #points_col[k][0] = points | points_col[k][1] =
col
```

```
        points_given=[] #list to visualise the given points per heuristic and the column played.
format-> list of lists = [[h_points,h_extra,col]]
        for i in range(len(childs)): col=childs[i][1]
            points_col.append([heuristic(state=(childs[i] [0]),col=col),col])
            #para visualizar pontuacao de cada heuristica

points_given.append([Game4InLine.heuristic_points((childs[i]
[0]),col),Game4InLine.heuristic_extra((childs[i][0]),col),col+1])

        points_col.sort() #order the list so that first is the lowest points in total and last the max
points. !!*this doesn't mean the best play is last*!!

        #para visualizar pontuacao de cada heuristica
        print(f"h_dada, h_extra, col:") for j in
        range(len(points_given)):
            print(points_given[j])
        #para visualizar pontuacao de cada heuristica

        return (points_col[0][1]) if self.pieces[self.turn] == 'O' else (points_col[-1][1])


    def ____str____(self): #override the print() method
        return print_board(self.board)



def getPoints(x,o):
    '''
    returns the points based on the given heuristic for the project '''
    if (x == 4 and o == 0): return 512
    if (x == 3 and o == 0): return 50
    if (x == 2 and o == 0): return 10
    if (x == 1 and o == 0): return 1
    if (x == 0 and o == 1): return -1
    if (x == 0 and o == 2): return -10
    if (x == 0 and o == 3): return -50
    if (x == 0 and o == 4):
```

```
            return -512
        return 0

def getPoints_extra(game: Game4InLine, x, o):
        '''
        returns the points based on the extra heuristic setup '''
        if(x==3 and o==1) and game.pieces[game.turn-1] == 'O': return -500
        if(x==2 and o==1) and game.pieces[game.turn-1] == 'O': return -100
        if(x==1 and o==3) and game.pieces[game.turn-1] == 'X': return 500
        if(x==1 and o==2) and game.pieces[game.turn-1] == 'X': return 100
        return 0

def print_board(board): #transform the game board from matrix to a visual representation
        board_str="|"
        for k in range(1,len(board[0])+1): board_str+=f"
                {k} |"
        board_str+="\n"
        for i in range(len(board)): board_str += "| "
                for j in range(len(board[i])):
                        #board_str+=board[i][j] board_str +=
                        f"{board[i][j]} | "
                board_str+="\n" return
        board_str
```

MCTS.py

Contém as class Node e MCTS

'MCTS' representa um árvore que usa 'Node' como estrutura de dados para as folhas da mesma

'MCTS' tem a lógica para o algoritmo Monte Carlo e a sua implementação

```
import random
import time import
math
from copy import deepcopy
from Game4InLine import Game4InLine


  class Node:
        '''
        class that defines the nodes for the MCTS tree '''
```

```
    def ____init_____(self, game: Game4InLine, parent=None):
        '''
        game is the game state, parent is the father node
        childs is started as empty so that we can differenciate from an explored node or no
        '''
        self.game=deepcopy(game)
        self.parent=parent self.visited=0
        self.wins=0
        self.childs = [] # formato [[node,col]]

    def set_childs(self):
        '''
        set the childs for the node based on the legal_moves from the state os the board
        appends a list to the list with the format ->  [child_node,
col]
        child_node is the possible game state from the current node
made by playing the column -> col '''
        poss_moves = self.game.legal_moves() for col in
        poss_moves:
            state = deepcopy(self.game) self.childs.append([Node(state.play(col),
parent=self),col])

    def UCB1(self):
        '''
        funcion that calculates the UCB1 for the node '''
        if self.visited == 0: return
            float('inf')

        return ((self.wins/self.visited) + math.sqrt(2) *
math.sqrt(2*math.log(self.parent.visited)/self.visited))

    def max_UCB(self):
        '''
        return the max UCB1 from the childs of the node
        used to know the best nodes at the explore/selection phase '''
        max_val = (self.childs[0][0]).UCB1() for i in range
        (1,len(self.childs)):
            max_val = max(max_val, (self.childs[i][0]).UCB1()) return max_val


class MCTS:
    '''
```

```
    class that defines a tree for MCTS '''

    def ____init_____(self, root: Game4InLine):
        self.root=Node(deepcopy(root))
        self.run_time = 0
        self.simulations = 0


    def search(self, time_limit, limit_simulations=22500):
        '''
        main function to execute the MCTS
        based on multiple tests, we found that MCTS works better with
5 seconds limit or 22500 iterations
        so we end the search when one of this conditions are broken '''
        start_time = time.time()
        simulations = 0
        while time.time() - start_time < time_limit and simulations < limit_simulations:
            node, col = self.selection()
            result = self.simulate(deepcopy(node.game),col) self.back_propagate(node,result)
            simulations += 1


        self.run_time = time.time() - start_time self.simulations =
        simulations


    def selection(self):
        '''
              we start on root node and will go to the childs of the node and select the best
   case to expand/simulate based on the UCB1 given and we repeat until we reach a leaf or
                                                                     the node as not been
visited
        if we can expand the node we select randomly from a childs '''
        node = self.root
        col_child = 0 # needed due to who we set up the isFinished()
funcion used in expand()

        while len(node.childs) != 0: max_ucb =
            node.max_UCB()
            max_nodes = [n for n in node.childs if n[0].UCB1() ==
max_ucb]
            best_child = random.choice(max_nodes) node =
            best_child[0]
            col_child = best_child[1]
```

```
                if node.visited == 0: return node

            if self.expand(node,col_child):
                random_child = random.choice(node.childs) node =
                random_child[0]

            return node

        def expand(self, node: Node, col: int):
            '''
            returns False if the game is over or True after we add childs if node is not a end for
            the game we add the childs
            '''
            if node.game.isFinished(col): return False

            node.set_childs() return True

        def simulate(self, node_state: Game4InLine, last_played: int):
            '''
            randomly select a valid column to play and repeats until the game is over
            return 0 if the game is lose or draw and 1 if win, based on the last piece played
            '''
            res = node_state.isFinished(last_played) #in case the node given is already finished
            if res:
                return res if res == 2 else 0 if node_state.turn%2==0 else
1

            while True:
                col = random.choice(node_state.legal_moves()) node_state.play(col)
                res = node_state.isFinished(col) if res:
                    return res if res == 2 else 0 if node_state.turn%2==0


else 1

        def back_propagate(self, node: Node, result):
            '''
            we go from the given node to the root of the tree and do the respective alterations to the
data
            when player 1 win we give to all the node where player 1 played +1 on wins and
+0 for else
            for when player 2 win we follow the same logic result = 2 if draw, =1 if

            'X' won and =0 if 'O' won
```

```
        logo quando result == node.game.turn vai dar reward 1 pois foi o jogador que
ganhou a simulacao
        '''
        while node != None:
            if result == node.game.turn: #when the winner is the same as the last player on
the curr node
                reward = 1 else:
            reward = 0 node.visited
            += 1 node.wins += reward
            node = node.parent


    def best_move(self):
        '''
        after the search() we select the best child from the root with this funcion
        we also print the win_rate for each child to visualize the data and the choise made
from the algoritm
        '''
        childs = self.root.childs if
        len(childs)==0:
            return random.choice(self.root.game.legal_moves()) node = childs[0][0]
        best_col = childs[0][1]
        max_win_rate = node.wins/node.visited
        print(f"win/visited: {max_win_rate:.4f} col: {best_col+1}")

        for i in range (1,len(childs)): node =
            childs[i][0]
            max_win_rate_temp = node.wins/node.visited print(f"win/visited:
            {max_win_rate_temp:.4f} col:
{childs[i][1]+1}")
            if    max_win_rate    <    max_win_rate_temp:
                max_win_rate    =    max_win_rate_temp
                best_col = childs[i][1]

        return   best_col   def

    statistic(self):
        ''' returns the number of simulations and the run time for the search taken '''
        return self.simulations, self.run_time
```

play.py

Tem como função ser a interface (pelo terminal) do jogo onde podemos escolher entre 3 modos Player vs Player | Player vs IA | IA vs IA (relativamente há IA pode escolher entre A* ou MCTS)

```
import time
from Game4InLine import Game4InLine

BOARD_SIZE_STANDARD = True #make it 'False' if u want to play 4InLine with a board diff from
6x7
TIME_MCTS = 5 #time for search with mcts

def result(game,col):
        '''
        funcao para analisar quando o jogo acabar se foi empate ou qual jogador ganhou
        '''
        res=game.isFinished(col) if res:
                if res==2:
                        print(f"Draw") else:
                        print(f"{game.pieces[game.turn-1]} won") return True
        return False

   def main():
          '''
        funcao para jogar Player vs Player | ou | Player vs IA
        permite configurar uma board diferente do 'normal' definido caso BOARD_SIZE_STANDARD
= False sendo a board minima de 5x5
        recebe um input y/n para saber se vai ser jogado contra IA e qual IA (A* ou MCTS)
        no main loop corremos o jogo, onde decidimos qual coluna jogar (column_played) por input
(players) ou decisao dos algoritmos (IA)
        loop acaba quando alguem ganhar ou empatar '''
        #in case user decides to play with a different board size from 6x7
        if BOARD_SIZE_STANDARD: game=Game4InLine(6,7) else:
                r,c=map(int,input("Min board size: 5x5\nBoard size: (rows,cols) ").split())
                if(r<=4 or c<=4): game=Game4InLine(6,7) else:
                game=Game4InLine(r,c)
        print(f"Board:\n{game}")

        #if user want to play vs AI
        Ai_playing = input("Play with AI [y\\n]: ") while Ai_playing!='y'
        and Ai_playing!='n':
                Ai_playing = input(f"Invalid choice\nPlay with AI [y\\n]: ")
        #and which AI [A* or MCTS]
        which_AI = 0
        if Ai_playing == "y":
            which_AI = int(input("A*: 1                        MCTS: 2\nChoose (1 or 2): ")) while
        which_AI != 1 and which_AI != 2:
```

```
        which_AI = int(input("Invalid choice\nA*: 1                    MCTS: 2\nChoose
(1 or 2): "))
    print("")


    #main loop
    while True:
        #player turn
        print(f"player {game.turn%2 +1} ('{game.pieces[game.turn%2]}')
turn")


        #Human play
        column_played = int(input("Column to place: ")) - 1 # -1 because the columns goes
from 0 to 6(or set col size) and user is expected to select a number from 1 to 7 (or set col size)
        while (column_played > game.cols-1 or column_played < 0) or
game.placed[column_played] >= game.rows :
            print("Impossible move")
            column_played = int(input("Column to place: ")) - 1

        game.play(column_played) print(f"Board:\n{game}")

        if result(game,column_played): break


        #AI play
        if Ai_playing == 'y':
            if which_AI == 1: #A*
                column_played=game.A_star(lambda state,col:
Game4InLine.heuristic_points(state,col)+ Game4InLine.heuristic_extra(state,col)) #A* requires a
heuristic as input and
                  game.play(column_played)
         # we use the given points heuritis and added a defensive heuristic
                     print(f"AI play: {column_played+1}")


            elif which_AI == 2: #MCTS tree =
                MCTS(game)
                tree.search(TIME_MCTS)
                column_played = tree.best_move() n_simulations, run_time=
                tree.statistic() game.play(column_played)
                print(f"AI play: {column_played+1}") print(f"Num simulations
                = {n_simulations} in
{run_time:.5f}seg")
```

```
                print(f"Board:\n{game}")
                if result(game,column_played): break

def main_A_star():
    '''
    this function was created to play A* vs A* without any human interaction
    as we have no variations to play, we will always play the same game/result
    and player 2 (O) is the winner all the time with 30 rounds played '''
    start_time = time.time() #timer to know how long it takes
    game=Game4InLine(6,7) #board is set to 'normal' size #main loop
    while True:
        print(f"AI {game.turn%2 +1} ('{game.pieces[game.turn%2]}') turn") # to know which
turn is X(1) or O(2)

        #play
        column_played=game.A_star(lambda state,col:
Game4InLine.heuristic_points(state,col)+ Game4InLine.heuristic_extra(state,col))
        game.play(column_played)
        print(f"AI play: {column_played+1}") print(f"Board:\n{game}")

        if result(game,column_played): break

    print(f"game took {(time.time()-start_time):.0f} seg and
{game.round} rounds")

def main_mcts():
    '''
    this funcion is made to play MCTS vs MCTS with out any human interaction
    '''
    start_time = time.time()
    game=Game4InLine(6,7) #main
    loop
    while True:
        print(f"AI {game.turn%2 +1} ('{game.pieces[game.turn%2]}')
turn")

        #AI play
        tree = MCTS(game) tree.search(TIME_MCTS)
        column_played = tree.best_move()
```

```
                n_simulations, run_time= tree.statistic()

                print(f"AI {game.turn%2 +1} play: {column_played+1}") print(f"Num simulations =
                {n_simulations} in
{run_time:.2f}seg")
                game.play(column_played)
                print(f"Board:\n{game}")

                if result(game,column_played): break

        print(f"game took {(time.time()-start_time):.0f} seg and
{game.round} rounds")

'''this is used in the file to select the type of game if name__ == "__main

______":
        #made so the user can choose what type of game he wants to play/watch
        qual = int(input(f"Qual modo:\n1: Player vs AI ou PvP\n2: A* vs A*\n3: MCTS vs
MCTS\nEscolha: "))
        if qual == 1: main()
        elif qual == 2: main_A_star()
        elif qual == 3: main_mcts()
'''
```

Resultado visual de um jogo com IA vs IA

A* vs A*

```
'''O resultado de A* vs A*
temos uma lista de 3 elementos [h_dada,h_extra,col] para visualizar qual a pontuação
atribuida por cada heuristica por coluna
*relembrar* o algoritmo tem em conta a soma da pontuação de cada '''
main_A_star()

AI 1 ('X') turn h_dada,
h_extra, col: [-13, 0, 1]
[-12, 0, 2]
[-11, 0, 3]
[-9, 0, 4]
[-11, 0, 5]
[-12, 0, 6]
[-13, 0, 7]
AI play: 4
```

```
Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | X | - | - | - |

AI 2 ('O') turn h_dada,
h_extra, col: [20, 0, 1]
[19, 0, 2]
[18, 0, 3]
[13, 0, 4]
[18, 0, 5]
[19, 0, 6]
[20, 0, 7]
AI play: 4 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | - | - | - |
| - | - | - | X | - | - | - |

AI 1 ('X') turn h_dada,
h_extra, col: [-8, 0, 1]
[1, 0, 2]
[10, 0, 3]
[-7, 0, 4]
[10, 0, 5]
[1, 0, 6]
[-8, 0, 7]
AI play: 5 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | - | - | - |
| - | - | - | X | X | - | - |

AI 2 ('O') turn h_dada,
h_extra, col: [39, 0, 1]
```

```
[29, -100, 2]
[11, -100, 3]
[22, 0, 4]
[10, 0, 5]
[20, -100, 6]
[30, -100, 7]
AI play: 3 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | - | - | - |
| - | - | O | X | X | - | - |

AI 1 ('X') turn h_dada,
h_extra, col: [-19, 0, 1]
[-19, 0, 2]
[-5, 0, 3]
[-9, 0, 4]
[3, 0, 5]
[21, 0, 6]
[21, 0, 7]
AI play: 7 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | - | - | - |
| - | - | O | X | X | - | X |

AI 2 ('O') turn h_dada,
h_extra, col: [51, 0, 1]
[51, 0, 2]
[13, 0, 3]
[33, 0, 4]
[21, 0, 5]
[1, -500, 6]
[41, 0, 7]
AI play: 6 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
```

```
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | - | - | - |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-29, 0, 1]
[-29, 0, 2]
[-15, 0, 3]
[-19, 0, 4]
[-7, 0, 5]
[-17, 0, 6]
[-19, 0, 7]
AI play: 5 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | X | - | - |
| - | - | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [23, 0, 1]
[23, 0, 2]
[3, 0, 3]
[6, 0, 4]
[-42, -100, 5]
[13, 0, 6]
[22, 0, 7]
AI play: 5 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | O | - | - |
| - | - | - | O | X | - | - |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-72, 0, 1]
[-72, 0, 2]
[-60, 0, 3]
[-55, 0, 4]
[-65, 0, 5]
```

```
[-71, 0, 6]
[-63, 0, 7]
AI play: 4 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | X | - | - |
| - | - | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [-25, 0, 1]
[-25, 0, 2]
[-45, 0, 3]
[-49, 0, 4]
[-40, 0, 5]
[-42, 0, 6]
[-26, 0, 7]
AI play: 4 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | - | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | X | - | - |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-79, 0, 1]
[-71, 0, 2]
[-51, 0, 3]
[-73, 0, 4]
[-48, 0, 5]
[-69, 100, 6]
[-70, 0, 7]
AI play: 6 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | - | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | X | X | - |
| - | - | O | X | X | O | X |
```

```
AI 2 ('O') turn h_dada,
h_extra, col: [-47, 0, 1]
[-39, 0, 2]
[-59, 0, 3]
[-45, 0, 4]
[-78, 0, 5]
[-51, -100, 6]
[-40, 0, 7]
AI play: 6 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | - | - | - |
| - | - | - | X | O | O | - |
| - | - | - | O | X | X | - |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-81, 0, 1]
[-73, 0, 2]
[-53, 0, 3]
[-75, 0, 4]
[-50, 0, 5]
[-19, 500, 6]
[-72, 0, 7]
AI play: 6 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | - | X | - |
| - | - | - | X | O | O | - |
| - | - | - | O | X | X | - |
| - | - | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [3, 0, 1]
[11, 0, 2]
[-9, 0, 3]
[-11, 0, 4]
[-26, 0, 5]
[9, 0, 6]
[2, 0, 7]
AI play: 5 Board:
```

```
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | O | X | - |
| - | - | - | X | O | O | - |
| - | - | - | O | X | X | - |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-56, 0, 1]
[-57, 0, 2]
[-46, 0, 3]
[-32, 100, 4]
[-34, 0, 5]
[-56, 0, 6]
[-38, 100, 7]
AI play: 4 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | X | - | - | - |
| - | - | - | O | O | X | - |
| - | - | - | X | O | O | - |
| - | - | - | O | X | X | - |
| - | - | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [-10, 0, 1]
[-1, 0, 2]
[-20, 0, 3]
[-6, 0, 4]
[-62, 0, 5]
[-2, 0, 6]
[-2, 0, 7]
AI play: 5 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | O | X | - |
| - | - | - | X | O | O | - |
| - | - | - | O | X | X | - |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-92, 0, 1]
[-93, 0, 2]
```

```
[-82, 0, 3]
[-89, 0, 4]
[-32, 500, 5]
[-94, 0, 6]
[-84, 0, 7]
AI play: 5 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | X | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | O | X | - |
| - | - | - | X | O | O | - |
| - | - | - | O | X | X | - |
| - | - | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [-10, 0, 1]
[-1, 0, 2]
[-20, 0, 3]
[-5, 0, 4]
[0, 0, 6]
[-2, 0, 7]
AI play: 3 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | X | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | O | X | - |
| - | - | - | X | O | O | - |
| - | - | O | O | X | X | - |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-50, 0, 1]
[-51, 0, 2]
[-9, 100, 3]
[-23, 0, 4]
[-52, 0, 6]
[-42, 0, 7]
AI play: 3 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | X | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | O | X | - |
| - | - | X | X | O | O | - |
| - | - | O | O | X | X | - |
```

```
| - | - | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [22, 0, 1]
[22, 0, 2]
[-49, -100, 3]
[18, 0, 4]
[23, 0, 6]
[21, 0, 7]
AI play: 3 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | X | - | - |
| - | - | - | X | O | - | - |
| - | - | O | O | O | X | - |
| - | - | X | X | O | O | - |
| - | - | O | O | X | X | - |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-80, 0, 1]
[-80, 0, 2]
[-61, 100, 3]
[-52, 0, 4]
[-81, 0, 6]
[-71, 0, 7]
AI play: 3 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | X | - | - |
| - | - | X | X | O | - | - |
| - | - | O | O | O | X | - |
| - | - | X | X | O | O | - |
| - | - | O | O | X | X | - |
| - | - | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [-30, 0, 1]
[-30, 0, 2]
[-32, 0, 3]
[-34, 0, 4]
[-29, 0, 6]
[-31, 0, 7]
AI play: 4 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | O | X | - | - |
```

```
| - | - | X | X | O | - | - |
| - | - | O | O | O | X | - |
| - | - | X | X | O | O | - |
| - | - | O | O | X | X | - |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-65, 0, 1]
[-65, 0, 2]
[-65, 0, 3]
[-66, 0, 6]
[-56, 0, 7]
AI play: 7 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | O | X | - | - |
| - | - | X | X | O | - | - |
| - | - | O | O | O | X | - |
| - | - | X | X | O | O | - |
| - | - | O | O | X | X | X |
| - | - | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [-25, 0, 1]
[-25, 0, 2]
[-33, 0, 3]
[-24, 0, 6]
[-36, -100, 7]
AI play: 7 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | O | X | - | - |
| - | - | X | X | O | - | - |
| - | - | O | O | O | X | - |
| - | - | X | X | O | O | O |
| - | - | O | O | X | X | X |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-67, 0, 1]
[-67, 0, 2]
[-67, 0, 3]
[-68, 0, 6]
[-67, 0, 7]
AI play: 7 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
```

```
| - | - | - | O | X | - | - |
| - | - | X | X | O | - | - |
| - | - | O | O | O | X | X |
| - | - | X | X | O | O | O |
| - | - | O | O | X | X | X |
| - | - | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [-36, 0, 1]
[-36, 0, 2]
[-44, 0, 3]
[-35, 0, 6]
[-35, 0, 7]
AI play: 3 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | O | O | X | - | - |
| - | - | X | X | O | - | - |
| - | - | O | O | O | X | X |
| - | - | X | X | O | O | O |
| - | - | O | O | X | X | X |
| - | - | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-75, 0, 1]
[-75, 0, 2]
[-76, 0, 6]
[-76, 0, 7]
AI play: 2 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | O | O | X | - | - |
| - | - | X | X | O | - | - |
| - | - | O | O | O | X | X |
| - | - | X | X | O | O | O |
| - | - | O | O | X | X | X |
| - | X | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [-44, 0, 1]
[-85, 0, 2]
[-43, 0, 6]
[-43, 0, 7]
AI play: 2 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | O | O | X | - | - |
```

```
| - | - | X | X | O | - | - |
| - | - | O | O | O | X | X |
| - | - | X | X | O | O | O |
| - | O | O | O | X | X | X |
| - | X | O | X | X | O | X |

AI 1 ('X') turn h_dada,
h_extra, col: [-116, 0, 1]
[-75, 0, 2]
[-117, 0, 6]
[-117, 0, 7]
AI play: 2 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | O | O | X | - | - |
| - | - | X | X | O | - | - |
| - | - | O | O | O | X | X |
| - | X | X | X | O | O | O |
| - | O | O | O | X | X | X |
| - | X | O | X | X | O | X |

AI 2 ('O') turn h_dada,
h_extra, col: [-44, 0, 1]
[-512, -512, 2]
[-43, 0, 6]
[-43, 0, 7]
AI play: 2 Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | O | O | X | - | - |
| - | - | X | X | O | - | - |
| - | O | O | O | O | X | X |
| - | X | X | X | O | O | O |
| - | O | O | O | X | X | X |
| - | X | O | X | X | O | X |

O won
game took 0 seg and 30 rounds
```

MCTS vs MCTS

```
''' Um resultado de MCTS vs MCTS
um resultado de diversos possiveis do algoritmo MCTS contra si mesmo e possivel visualizar
qual das colunas possiveis de jogar tem a maior
% de vitoria apos o algoritmo correr qual foi a
escolhida
e o numero de iteracoes e o tempo que demorou
```

```
'''
main_mcts()

AI 1 ('X') turn win/visited: 0.5265 col:
1
win/visited: 0.5326 col: 2
win/visited: 0.5533 col: 3
win/visited: 0.6396 col: 4
win/visited: 0.5563 col: 5
win/visited: 0.5298 col: 6
win/visited: 0.4709 col: 7 AI 1 play: 4
Num simulations = 14893 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | X | - | - | - |

AI 2 ('O') turn win/visited: 0.3178
col: 1
win/visited: 0.3201 col: 2
win/visited: 0.3886 col: 3
win/visited: 0.4018 col: 4
win/visited: 0.3752 col: 5
win/visited: 0.3302 col: 6
win/visited: 0.2800 col: 7 AI 2 play: 4
Num simulations = 14593 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | O | - | - | - |
| - | - | - | X | - | - | - |

AI 1 ('X') turn win/visited: 0.5509 col:
1
win/visited: 0.5634 col: 2
win/visited: 0.5970 col: 3
win/visited: 0.6139 col: 4
win/visited: 0.6118 col: 5
win/visited: 0.5659 col: 6
win/visited: 0.5567 col: 7 AI 1 play: 4
```

```
Num simulations = 15433 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | X | - | - | - |
| - | - | - | O | - | - | - |
| - | - | - | X | - | - | - |

AI 2 ('O') turn win/visited: 0.3636
col: 1
win/visited: 0.4134 col: 2
win/visited: 0.4035 col: 3
win/visited: 0.3903 col: 4
win/visited: 0.4153 col: 5
win/visited: 0.3718 col: 6
win/visited: 0.3512 col: 7 AI 2 play: 5
Num simulations = 15501 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | X | - | - | - |
| - | - | - | O | - | - | - |
| - | - | - | X | O | - | - |

AI 1 ('X') turn win/visited: 0.5644 col:
1
win/visited: 0.5733 col: 2
win/visited: 0.5409 col: 3
win/visited: 0.5893 col: 4
win/visited: 0.6242 col: 5
win/visited: 0.4994 col: 6
win/visited: 0.5119 col: 7 AI 1 play: 5
Num simulations = 15401 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | X | - | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |

AI 2 ('O') turn win/visited: 0.3550 col:
1
```

```
win/visited: 0.3641 col: 2
win/visited: 0.3329 col: 3
win/visited: 0.3728 col: 4
win/visited: 0.4045 col: 5
win/visited: 0.3145 col: 6
win/visited: 0.3090 col: 7 AI 2 play: 5
Num simulations = 15610 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |

AI 1 ('X') turn win/visited: 0.5333 col:
1
win/visited: 0.5843 col: 2
win/visited: 0.5655 col: 3
win/visited: 0.6348 col: 4
win/visited: 0.5980 col: 5
win/visited: 0.5279 col: 6
win/visited: 0.5489 col: 7 AI 1 play: 4
Num simulations = 15762 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | - | - | - |
| - | - | - | X | - | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |

AI 2 ('O') turn win/visited: 0.3490
col: 1
win/visited: 0.3845 col: 2
win/visited: 0.3526 col: 3
win/visited: 0.3901 col: 4
win/visited: 0.4047 col: 5
win/visited: 0.3371 col: 6
win/visited: 0.2711 col: 7 AI 2 play: 5
Num simulations = 15807 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
```

```
| - | - | - | - | - | - | - |
| - | - | - | X | O | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |

AI 1 ('X') turn win/visited: 0.5675 col:
1
win/visited: 0.5389 col: 2
win/visited: 0.6084 col: 3
win/visited: 0.5865 col: 4
win/visited: 0.6247 col: 5
win/visited: 0.5072 col: 6
win/visited: 0.5174 col: 7 AI 1 play: 5
Num simulations = 15817 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | - | X | - | - |
| - | - | - | X | O | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |

AI 2 ('O') turn win/visited: 0.3606
col: 1
win/visited: 0.3759 col: 2
win/visited: 0.3753 col: 3
win/visited: 0.4375 col: 4
win/visited: 0.2870 col: 5
win/visited: 0.3600 col: 6
win/visited: 0.3247 col: 7 AI 2 play: 4
Num simulations = 16836 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |

AI 1 ('X') turn win/visited: 0.5652 col:
1
win/visited: 0.5695 col: 2
win/visited: 0.6252 col: 3
win/visited: 0.5153 col: 4
win/visited: 0.5029 col: 5
```

```
win/visited: 0.5123 col: 6
win/visited: 0.5552 col: 7 AI 1 play: 3
Num simulations = 17456 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | X | - | - |
| - | - | X | X | O | - | - |

AI 2 ('O') turn win/visited: 0.4165
col: 1
win/visited: 0.4352 col: 2
win/visited: 0.3084 col: 3
win/visited: 0.3093 col: 4
win/visited: 0.3205 col: 5
win/visited: 0.3920 col: 6
win/visited: 0.3125 col: 7 AI 2 play: 2
Num simulations = 16386 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |
| - | - | - | X | O | - | - |
| - | - | - | O | X | - | - |
| - | O | X | X | O | - | - |

AI 1 ('X') turn win/visited: 0.5084 col:
1
win/visited: 0.6368 col: 2
win/visited: 0.5483 col: 3
win/visited: 0.5222 col: 4
win/visited: 0.5053 col: 5
win/visited: 0.5521 col: 6
win/visited: 0.5199 col: 7 AI 1 play: 2
Num simulations = 17774 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |
| - | - | - | X | O | - | - |
| - | X | - | O | X | - | - |
```

```
| - | O | X | X | O | - | - |

AI 2 ('O') turn win/visited: 0.3450
col: 1
win/visited: 0.3618 col: 2
win/visited: 0.3099 col: 3
win/visited: 0.3498 col: 4
win/visited: 0.3402 col: 5
win/visited: 0.3960 col: 6
win/visited: 0.3391 col: 7 AI 2 play: 6
Num simulations = 18026 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |
| - | - | - | X | O | - | - |
| - | X | - | O | X | - | - |
| - | O | X | X | O | O | - |

AI 1 ('X') turn win/visited: 0.5849 col:
1
win/visited: 0.6107 col: 2
win/visited: 0.4953 col: 3
win/visited: 0.6089 col: 4
win/visited: 0.6081 col: 5
win/visited: 0.5390 col: 6
win/visited: 0.5784 col: 7 AI 1 play: 2
Num simulations = 18560 in 5.00seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | O | X | - | - |
| - | - | - | X | O | - | - |
| - | X | - | X | O | - | - |
| - | X | - | O | X | - | - |
| - | O | X | X | O | O | - |

AI 2 ('O') turn win/visited: 0.2870
col: 1
win/visited: 0.3946 col: 2
win/visited: 0.2600 col: 3
win/visited: 0.2912 col: 4
win/visited: 0.2660 col: 5
win/visited: 0.3348 col: 6
win/visited: 0.3235 col: 7 AI 2 play: 2
Num simulations = 20115 in 5.00seg
```

```
Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | O | X | - | - |
| - | O | - | X | O | - | - |
| - | X | - | X | O | - | - |
| - | X | - | O | X | - | - |
| - | O | X | X | O | O | - |

AI 1 ('X') turn win/visited: 0.5078 col:
1
win/visited: 0.4991 col: 2
win/visited: 0.6445 col: 3
win/visited: 0.5123 col: 4
win/visited: 0.4828 col: 5
win/visited: 0.4995 col: 6
win/visited: 0.5082 col: 7 AI 1 play: 3
Num simulations = 22500 in 4.05seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | O | X | - | - |
| - | O | - | X | O | - | - |
| - | X | - | X | O | - | - |
| - | X | X | O | X | - | - |
| - | O | X | X | O | O | - |

AI 2 ('O') turn win/visited: 0.2744
col: 1
win/visited: 0.2434 col: 2
win/visited: 0.3863 col: 3
win/visited: 0.2454 col: 4
win/visited: 0.2312 col: 5
win/visited: 0.2784 col: 6
win/visited: 0.2609 col: 7 AI 2 play: 3
Num simulations = 22500 in 2.79seg Board:
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| - | - | - | - | - | - | - |
| - | - | - | O | X | - | - |
| - | O | - | X | O | - | - |
| - | X | O | X | O | - | - |
| - | X | X | O | X | - | - |
| - | O | X | X | O | O | - |

O won
game took 87 seg and 18 rounds
```