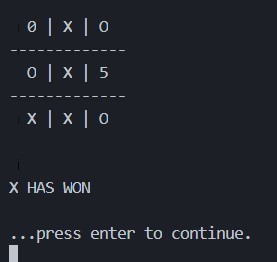
Thomas Mulvey

Project 1.A : TTT WITH MINIMAX

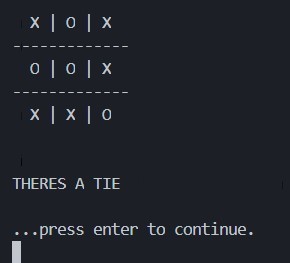
4/1/19

Analysis / Sample Outputs: “X” is COMPUTER and goes first

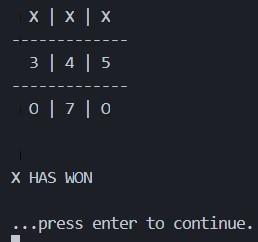
Game 1: HUMAN LOSS



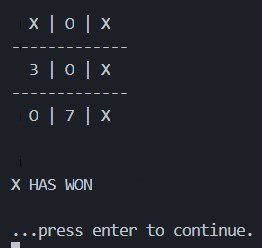
Game 2: TIE



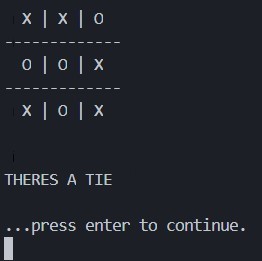
Game 3: HUMAN LOSS



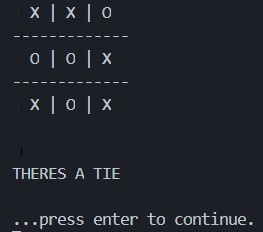
Game 4: HUMAN LOSS



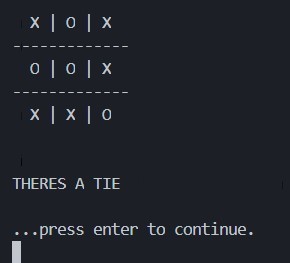
Game 5: TIE



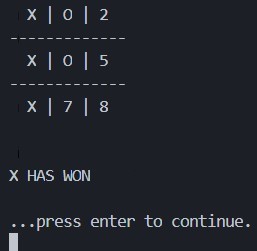
Game 6: TIE



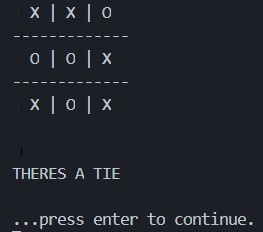
Game 7: TIE



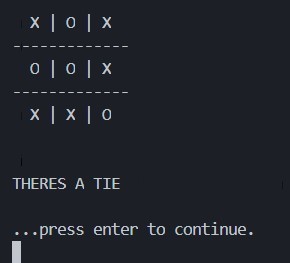
Game 8: HUMAN LOSS



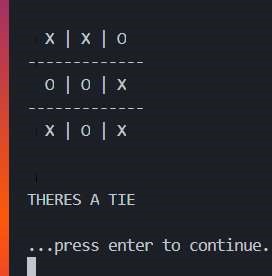
Game 9: TIE



Game 10: TIE



MINIMAX vs MINIMAX: TIE



Results:

Humans vs AI (Human Wins, AI Win, TIE) -- (0, 4, 6)

* Game 3 and 8: AI obviously sees optimal path
* Game 4 and 1: Starting a corner like the AI results in loss
* Starting in middle after corner move is only move that can result in a tie

AI vs AI: always a tie

**== CODE ==**

**MINIMAX CLASS:**

import random

from Player import \*

# X IS MAX (computer) O IS MINS (human)

# we will return score based off of X's position

class MiniMax(Player):

def \_\_init\_\_(self, char='X'):

self.char = char

self.kind = 'MiniMax'

if self.char == 'X':

self.opponent = 'O'

else :

self.opponent = 'X'

'''

is game done given board state?

returns (TRUE, value of state [10 win, -10 lost] ) or FALSE

'''

def is\_terminal\_state(self, board):

winning\_states = ( [0,1,2],[3,4,5],[6,7,8],[0,3,6],[1,4,7],[2,5,8],[0,4,8],[2,4,6] )

for a,b,c in winning\_states:

if board[a]==board[b]==board[c]==self.char:

return (True, 10) #minimax won!

elif board[a]==board[b]==board[c]==self.opponent:

return (True, -10) #other player won

space\_counter = 0

for spot in board:

if spot=='█':

space\_counter+=1

if space\_counter==0: #TIE

return (True, 0)

return (False, 0) # aint over yet, chiefton

def move(self, board): #acutal MINIMAX IMPLEMENTATION

# in order to cut down brnaching factor a bit, IF ai

# is going first, just choose a corner.

if len( self.available\_positions(board) ) == 9:

return random.choice( [0,2,6,8] ) , 10

# ON THE MINIMAX TURN, YOU WANT THE BEST (MAX) OF THE OTHER PLAYERS TURNS(MIN)

moves=[-10 for \_ in range(9)] #move values

for move in self.available\_positions(board) : # for every child, is it a winner? is a successor a winner? else play random

board[int(move)] = str(self.char)

if (self.is\_terminal\_state(board))[0] is True:

return move, (self.is\_terminal\_state(board))[1]

board\_val = self.min\_value(board)

board[move] = '█'

moves[move] = board\_val

# game would sometimes choose a slower win so this forces it to choose immediate win

c=0

for i in moves:

if i == 0 and board[c] == '█':

board[c] = self.opponent

res = (self.is\_terminal\_state(board))[1]

if int(res) == int(-10):

return c,0 #stops win so tie for now

board[c] = '█'

c+=1

# otherwise play random move

return moves.index(max(moves)) , max(moves)

# if cant find a move there, just take a tie from here.

# return random.choice(self.available\_positions(board))

def max\_value(self, board):

board\_done, return\_value = self.is\_terminal\_state(board)

if board\_done: # if current board is done, return -10, 0 , 10

return return\_value

value = -100

for moves in self.available\_positions(board):

board[moves] = self.char

new\_value = self.min\_value(board)

if new\_value > value:

value = new\_value

board[moves] = '█'

return value

def min\_value(self, board):

board\_done, return\_value = self.is\_terminal\_state(board)

if board\_done:

return return\_value

value = 100

for moves in self.available\_positions(board):

board[moves] = self.opponent

new\_value = self.max\_value(board)

if new\_value < value:

value = new\_value

board[moves] = '█'

return value

**PLAYER / HUMAN CLASS**

class Player:

def \_\_init\_\_(self, char='X'):

self.kind = 'human'

self.char = char

def move(self, board):

while True: #valid move

move = int(input('Your move? '))

if board[move] != "X" and board[move] != "O" and move >= 0 and move <= 9:

return move

def available\_positions(self, board):

return [i for i in range(0, 9) if board[i] == '█']

**FOOTNOTE:** I forgot to add the values associated with each move in the screenshot so here is minimax vs minimax where it displays it.

