Ministerul Educaţiei, al Culturii și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Departamentul Informatică și Ingineria Sistemelor

**RAPORT**

Lucrarea de laborator nr.3

Inteligenta Artificiala

A efectuat:

st. gr. C-171 D. Melniciuc

A verificat:

dr., conf.univ. T. Bumbu

Chişinău 2020

Jocul ales:

Tic-Tac-Toe

Codul:

from math import inf

import sys, os

import numpy as np

HUMAN = 1

COMP = -1

board = [[0, 0, 0],

[0, 0, 0],

[0, 0, 0]]

def evaluate(state):

if wins(state, COMP):

score = -1

elif wins(state, HUMAN):

score = 1

else:

score = 0

return score

def empty\_cells(state):

cells = [] # it contains all empty cells

for i, row in enumerate(state):

for j, col in enumerate(row):

if state[i][j] == 0:

cells.append([i, j])

return cells

def wins(state, player):

win\_state = [

[state[0][0], state[0][1], state[0][2]],

[state[1][0], state[1][1], state[1][2]],

[state[2][0], state[2][1], state[2][2]],

# vertical

[state[0][0], state[1][0], state[2][0]],

[state[0][1], state[1][1], state[2][1]],

[state[0][2], state[1][2], state[2][2]],

# diagonal

[state[0][0], state[1][1], state[2][2]],

[state[2][0], state[1][1], state[0][2]],

]

if [player, player, player] in win\_state:

return True

else:

return False

def game\_over(state):

return wins(state, HUMAN) or wins(state, COMP)

def minimax(state, depth, player):

if player == COMP:

best = [-1, -1, inf] # inf/-inf are the initial score for the players

else:

best = [-1, -1, -inf]

if depth == 0 or game\_over(state):

score = evaluate(state)

return [-1, -1, score]

for cell in empty\_cells(state):

# Fill the empty cells with the player symbols

x, y = cell[0], cell[1]

state[x][y] = player

score = minimax(state, depth - 1, -player)

state[x][y] = 0

score[0], score[1] = x, y

if player == COMP:

if score[2] < best[2]:

best = score

else:

if score[2] > best[2]:

best = score

return best

def human\_turn(state):

# All possible moves

moves = {

1: [0, 0], 2: [0, 1], 3: [0, 2],

4: [1, 0], 5: [1, 1], 6: [1, 2],

7: [2, 0], 8: [2, 1], 9: [2, 2],

}

remain = empty\_cells(state)

isTurn = True

while isTurn:

try:

move = int(input("select pos (1-9): "))

if moves.get(move) in remain:

x, y = moves.get(move)

state[x][y] = HUMAN

isTurn = False

else:

print("ocupied")

except ValueError:

print("input is not a number!")

# While-else loop, this code below will run after successful loop.

else:

os.system("cls")

render(state)

def ai\_turn(state):

depth = len(empty\_cells(state)) # The remaining of empty cells

row, col, score = minimax(state, depth, COMP) # the optimal move for computer

state[row][col] = COMP

print("A.I Turn")

render(state) # Show result board

arr2 = [None] \* 10

def render(state):

"""Render the board state to stdout"""

legend = {0: " ", 1: "X", -1: "O"}

state = list(map(lambda x: [legend[y] for y in x], state))

result = "{}\n{}\n{}\n".format(\*state)

arr2 = np.array(state)

arr = arr2.flatten()

print("\n")

print(' ' , arr[6] , " | " , arr[7] , " | " , arr[8] , "\t 7 | 8 | 9")

print(' ' , arr[3] , " | " , arr[4] , " | " , arr[5] , "\t 4 | 5 | 6")

print(' ' , arr[0] , " | " , arr[1] , " | " , arr[2] , "\t 1 | 2 | 3")

print("\n")

return result

def main():

os.system("cls")

print("\n")

print(" | | " , " \t 7 | 8 | 9")

print(" | | " , " \t 4 | 5 | 6")

print(" | | " , " \t 1 | 2 | 3\n")

while not wins(board, COMP) and not wins(board, HUMAN):

human\_turn(board)

if len(empty\_cells(board)) == 0: break

ai\_turn(board)

if wins(board, COMP):

print("AI wins")

elif wins(board, HUMAN):

print("imposible")

else:

print("tie")

main()

while True:

op = input("\nx to exit: ")

if op.lower() == "x":

exit(0)

else:

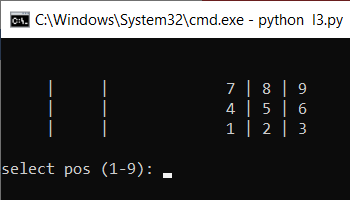
board = [[0, 0, 0],

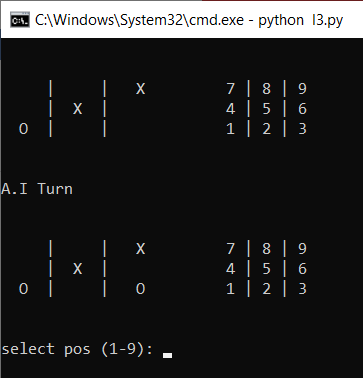
[0, 0, 0],

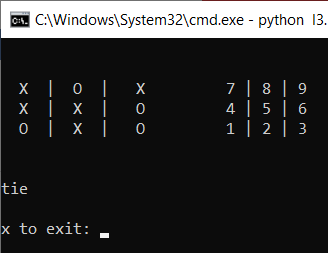
[0, 0, 0]]

main()

Screenshot-uri:







Concluzie:

In urma efectuarii laboratorului au elaborat un algoritm determinist minimax pentru jocul Tic-Tac-Toe. Am obeservat performantele acestui algoritm, el nu poate fi cistigat. Primul pas al jucatorului AI dureaza putin din cauza crearii arborelor de miscari si trecerea prin acesta pentru a gasi pasul cel mai bun.