„Monte Carlo” dla „Tic Tac Toe”

# Autorzy:

* Vladyslav Kubyshkin

# Opis:

Cel projektu – demonstracja działania algorytmu „Monte Carlo” dla gry „Tic Tac Toe”. Projekt napisany na języku programowania Python.

# Kod:

import random

# wyświetlamy tablicę wiersz po wierszu, można to zrobić w pętlach, ale tak czytelniej

def print\_board(board):

    print(f"{board[0][0]}|{board[0][1]}|{board[0][2]}")

    print(f"{board[1][0]}|{board[1][1]}|{board[1][2]}")

    print(f"{board[2][0]}|{board[2][1]}|{board[2][2]}")

    print()

def check\_input(userInput):   # tu sprawdzamy input w dwóch funkcjach

    if not isnum(userInput):

        # czy jest to liczba

        return False

    userInput = int(userInput)   # konwertujemy do integera

    if not bounds(userInput):

        # czy jest to potrzebna liczba

        return False

    return True

def isnum(userInput):   # tu sprawdzamy, czy input jest numerem

    if not userInput.isnumeric():

        print("This is not a valid number")

        return False

    else:

        return True

def bounds(userInput):   # tu sprawdzamy, czy numer jest w zakresie od 1 do 9

    if userInput > 9 or userInput < 1:

        print("This is number is out of bounds")

        return False

    else:

        return True

def istaken(coords, board):   # tu sprawdzamy, czy komórka zajęta

    row = coords[0]

    col = coords[1]

    if board[row][col] != "-":

        print("This position is already taken.")

        return True

    else:

        return False

# najtrudniejsza funkcja skryptu, powiecmy input 5, to po odejmowaniu o 1 dostajemy 4

def \_convertIndexToCords(userInput):

    row = int(userInput / 3)   # 4 / 3 = 1, matematyka nie zaakceptuje

    col = userInput   # jeżeli input 0 lub 1 lub 2, to kolumna będzie taka sama

    if col > 2:

        # jeżeli większy robimy mod 3

        col = int(col % 3)   # 4 % 3 = 1

    return (row, col)   # zwracamy 2d indexy dla 2d listy

def add\_to\_board(coords, board, activeChar):   # dodajemy znak do deski

    board[coords[0]][coords[1]] = activeChar

def iswin(xMove, board):

    if check\_row(xMove, board):

        # sprawdzamy, czy wygrana po wierszach

        return True

    if check\_col(xMove, board):

        # sprawdzamy, czy wygrana po kolumnach

        return True

    if check\_diag(xMove, board):

        # sprawdzamy, czy wygrana po kolumnach

        return True

    return False

def check\_row(xMove, board):   # sprawdzamy, czy wygrana po wierszach

    for row in board:

        complete\_row = True

        for slot in row:

            if slot != xMove:

                complete\_row = False

                break

        if complete\_row:

            return True

    return False

def check\_col(xMove, board):   # sprawdzamy, czy wygrana po kolumnach

    for col in range(3):

        complete\_col = True

        for row in range(3):

            if board[row][col] != xMove:

                complete\_col = False

                break

        if complete\_col:

            return True

    return False

def check\_diag(xMove, board):   # sprawdzamy, czy wygrana po przekątnych

    if board[0][0] == xMove and board[1][1] == xMove and board[2][2] == xMove:

        return True

    elif board[0][2] == xMove and board[1][1] == xMove and board[2][0] == xMove:

        return True

    else:

        return False

def bot\_move():   # tu odbywa się logika działania bota

    i = 0   # init dla i

    freeCells = []   # tworzymy pustą listę

    while i < 9:          # sprawdzamy wszystkie komórki po kolei

        # konwertujemy index do 2d indexu

        cordsBot = \_convertIndexToCords(i)

        if board[cordsBot[0]][cordsBot[1]] == "-":

            # sprawdzamy, czy jest pusta

            freeCells.append(i)   # jeżeli pusta dodajemy do listy

        i += 1   # i++;

    # w tym wierszu widzimy czemu python jest popularniejszym od innych języków programowania

    return random.choice(freeCells)

board = [

    ["-", "-", "-"],

    ["-", "-", "-"],

    ["-", "-", "-"]

]   # tworzymy deskę do grania, gdzie - to jest puste pole

xMove = True   # czy teraz ruch gracza znaku X

turn = 1   # numer ruchu potrzebny do sprawdzania remisu

charChoosen = "x"   # init dla znaku, który wybierze grać

while True:

    # pętla dla wybrania graczem znaku

    # tu prosimy o wpisywanie znaku i pobierany dane od gracza

    charChoosen = input("Choose your char (x/o): ")

    if charChoosen.lower() == "o" or charChoosen.lower() == "x":

        # sprawdzamy, czy jest znak odpowiedni

        # konwertujemy w "lower case" dla poprawnego działania skryptu

        charChoosen = charChoosen.lower()

        break   # wyłączymy pętle

    else:

        # informujemy, że znak zły i pętla włączy się jeszcze raz

        print("Looks like bad char, please choose between x and o.")

while turn <= 9:

    # pętla działa do 9 ruchu, bo tyle mamy komórek na desce

    userInput = ""   # pusty init dla userInput

    activeChar = ""   # pusty init dla activeChar

    # przypisujemy znak, który będzie robił teraz ruch

    if xMove:

        activeChar = "x"

    else:

        activeChar = "o"

    if activeChar == charChoosen:

        # sprawdzamy, czy jest znak, który chodzi teraz był wybrany przez gracza

        print\_board(board)   # wyśletliamy boisko

        userInput = input(

            "Please enter a position 1 through 9 or enter \"q\" for exit: ")   # pobieramy input indexa od gracza

        if userInput.lower() == "q":

            # jeżeli wprowadzono q wyłączamy pętle

            print("Thanks for playing")   # grzecznie żegnamy się

            break   # wyłączamy pętle

        if not check\_input(userInput):

            # sprawdzamy input

            print("Please try again.")

            continue

        # jeżeli input się zgadza, przypisujemy do zmiennej i odejmujemy 1

        userInput = int(userInput) - 1

        # konwertujemy index do 2d listy, 1 = 0|0, 9 = 2|2 i td.

        coords = \_convertIndexToCords(userInput)

        if istaken(coords, board):

            # sprawdzamy, czy jest wolne

            print("Please try again. Field not free")

            continue

    else:

        # jeżeli znak nie odpowiada temu, który wybrał grać to ruch robi bot

        # ponieważ wiemy, że robot nie jest na tyle inteligenty, żeby wprowadzić niepoprawne dane, jak by to zrobił mądry użytkownik, nie potrzebujemy tyle sprawdzeń

        userInput = int(bot\_move())

        # konwertujemy output robota do 2d listy, 1 = 0|0, 9 = 2|2 i td.

        coords = \_convertIndexToCords(userInput)

    # dodajemy znak do tablicy

    board[coords[0]][coords[1]] = activeChar

    # sprawdzamy, czy ktoś jakimś cudem nie wygrał

    if iswin(activeChar, board):

        print\_board(board)

        print(f"{activeChar.upper()} won!")

        break

    turn += 1

    if turn == 10:

        # informujemy o remisie

        print("Tie!")

    xMove = not xMove   # przekazujemy ruch innemu znaku