TIC TAC TOE GAME

AI Assignment



**GITHUB LINK -**

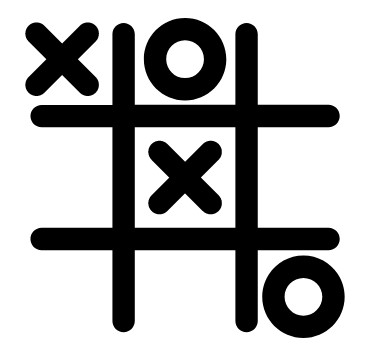
https://github.com/Divyansh-Ag/AI/tree/master

Under the Supervision of

Mr. MOIN HASAN

Submitted By:

1. DIVYANSH AGRAWAL (11808716), Roll no. - 64
2. NIGAM BHATTARIA (11813477), Roll no. - 57
3. PANKAJ SINGH KOSHYARI (11813439), Roll no. - 65



Abstract:

Our project name is Tic-Tac-Toe game. This game is very popular and is fairly simple by itself. It is actually a two player game. In this game, there is a board with *n* x *n* squares. In our game, it is 3 x 3 squares. The goal of Tic-Tac-Toe is to be one of the players to get three same symbols in a row - horizontally, vertically or diagonally - on a 3 x 3 grid.

Introduction:

Rules of the Game:

• The game is to be played between two people (in this program between HUMAN and

COMPUTER).

• One of the player chooses ‘O’ and the other ‘X’ to mark their respective cells.

• The game starts with one of the players and the game ends when one of the players has

One whole row/ column/ diagonal filled with his/her respective character (‘O’ or ‘X’).

A player can play perfect tic-tac-toe (win or draw) given they move according to the highest

Possible move from the following table.

**1. Win:** If the player has two in a row, play the third to get three in a row.

**2. Block:** If the opponent has two in a row, play the third to block them.

**3. Fork:** Create an opportunity where you can win in two ways

**4. Block opponent's fork:** *a. Option 1:* Create two in a row to force the opponent into defending, as long as it doesn’t result in them creating a fork or winning. For example, if "X" has a corner, "O" has the center, and "X" has the opposite corner as well, "O" must not play a Corner in order to win. (Playing a corner in this scenario creates a fork for "X" to win).

*b. Option 2:* If there is a configuration where the opponent can fork, block that fork.

**5. Center:** Play the center.

**6. Opposite corner:** If the opponent is in the corner, play the opposite corner.

**7. Empty corner:** Play in a corner square.

**8. Empty side:** Play in a middle square on any of the 4 sides.

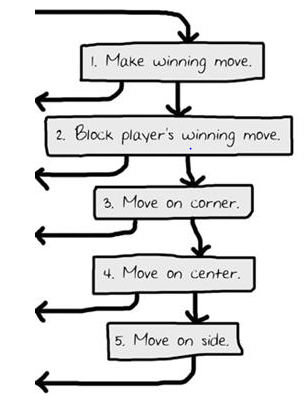
## Game Strategy:

Tic-Tac-Toe game has many strategies that can be used. The main point of the strategy is the players have to block the opponent fork, either horizontally, vertically, or diagonally, while the players have to find their own fork to win.

In combinatorial study, suppose “X” moves first, then the game is won as follows:

* 91 distinct positions are won by X.
* 44 distinct position are won by O.
* 3 distinct positions are draw.

Steps Required By the Computer:



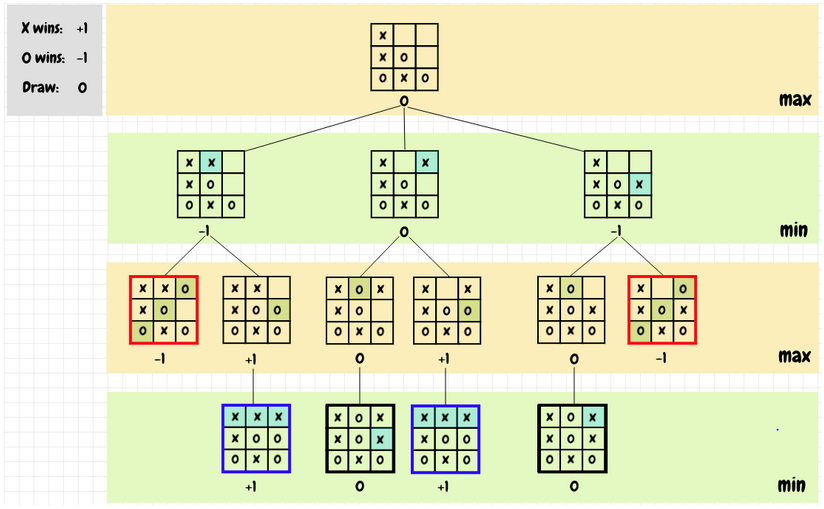
Algorithm Used:

Minimax Algorithm –

Minimax is a recursive algorithm which is used to choose an optimal move for a player assuming that the opponent is also playing optimally. As its name suggests, its goal is to minimize the maximum loss (minimize the worst case scenario).

Advantages of using Minimax Algorithm –

* Best suited for game with two players.
* Finds a path from starting to goal position.
* Calculate all possible game states by examining all opposing moves.
* Determine the next move against best play [opponent].



Game Tree of TIC TAC TOE

Code:

from tkinter import Tk, Button

from tkinter.font import Font

from copy import deepcopy

class Board:

def \_\_init\_\_(self,other=None):

self.player = 'X'

self.opponent = 'O'

self.empty = '.'

self.size = 3

self.fields = {}

for y in range(self.size):

for x in range(self.size):

self.fields[x,y] = self.empty

# copy constructor

if other:

self.\_\_dict\_\_ = deepcopy(other.\_\_dict\_\_)

def move(self,x,y):

board = Board(self)

board.fields[x,y] = board.player

(board.player,board.opponent) = (board.opponent,board.player)

return board

def \_\_minimax(self, player):

if self.won():

if player:

return (-1,None)

else:

return (+1,None)

elif self.tied():

return (0,None)

elif player:

best = (-2,None)

for x,y in self.fields:

if self.fields[x,y]==self.empty:

value = self.move(x,y).\_\_minimax(not player)[0]

if value>best[0]:

best = (value,(x,y))

return best

else:

best = (+2,None)

for x,y in self.fields:

if self.fields[x,y]==self.empty:

value = self.move(x,y).\_\_minimax(not player)[0]

if value<best[0]:

best = (value,(x,y))

return best

def best(self):

return self.\_\_minimax(True)[1]

def tied(self):

for (x,y) in self.fields:

if self.fields[x,y]==self.empty:

return False

return True

def won(self):

# horizontal

for y in range(self.size):

winning = []

for x in range(self.size):

if self.fields[x,y] == self.opponent:

winning.append((x,y))

if len(winning) == self.size:

return winning

# vertical

for x in range(self.size):

winning = []

for y in range(self.size):

if self.fields[x,y] == self.opponent:

winning.append((x,y))

if len(winning) == self.size:

return winning

# diagonal

winning = []

for y in range(self.size):

x = y

if self.fields[x,y] == self.opponent:

winning.append((x,y))

if len(winning) == self.size:

return winning

# other diagonal

winning = []

for y in range(self.size):

x = self.size-1-y

if self.fields[x,y] == self.opponent:

winning.append((x,y))

if len(winning) == self.size:

return winning

# default

return None

def \_\_str\_\_(self):

string = ''

for y in range(self.size):

for x in range(self.size):

string+=self.fields[x,y]

string+="\n"

return string

class GUI:

def \_\_init\_\_(self):

self.app = Tk()

self.app.title('TicTacToe')

self.app.resizable(width=False, height=False)

self.board = Board()

self.font = Font(family="Helvetica", size=32)

self.buttons = {}

for x,y in self.board.fields:

handler = lambda x=x,y=y: self.move(x,y)

button = Button(self.app, command=handler, font=self.font, width=2, height=1)

button.grid(row=y, column=x)

self.buttons[x,y] = button

handler = lambda: self.reset()

button = Button(self.app, text='reset', command=handler)

button.grid(row=self.board.size+1, column=0, columnspan=self.board.size, sticky="WE")

self.update()

def reset(self):

self.board = Board()

self.update()

def move(self,x,y):

self.app.config(cursor="watch")

self.app.update()

self.board = self.board.move(x,y)

self.update()

move = self.board.best()

if move:

self.board = self.board.move(\*move)

self.update()

self.app.config(cursor="")

def update(self):

for (x,y) in self.board.fields:

text = self.board.fields[x,y]

self.buttons[x,y]['text'] = text

self.buttons[x,y]['disabledforeground'] = 'black'

if text==self.board.empty:

self.buttons[x,y]['state'] = 'normal'

else:

self.buttons[x,y]['state'] = 'disabled'

winning = self.board.won()

if winning:

for x,y in winning:

self.buttons[x,y]['disabledforeground'] = 'red'

for x,y in self.buttons:

self.buttons[x,y]['state'] = 'disabled'

for (x,y) in self.board.fields:

self.buttons[x,y].update()

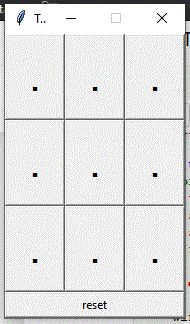
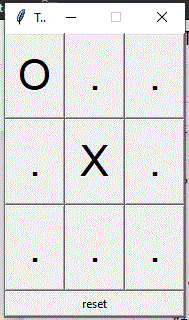
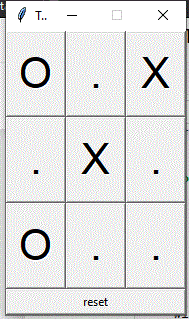
def mainloop(self):

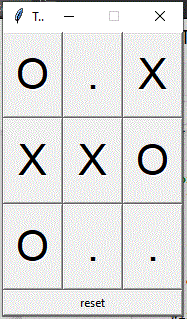
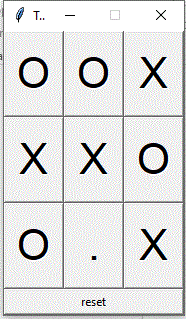
self.app.mainloop()

if \_\_name\_\_ == '\_\_main\_\_':

GUI().mainloop()

Result & Conclusion:

DRAW

Due to the relatively small state space (3⁹ = 196839 possible board combinations), it can easily search the whole game tree for an optimal solution, treating the game as a fully deterministic environment.

Making it **Unbeatable.**

**Tic Tac Toe AI** is unbeatable. You can draw at most and only with a perfect game. If you think that you can outsmart it and win the game, than try to do.

References:

* Google.
* Artificial Intelligence: A Modern Approach.
* Artificial Intelligence With Python.