Code

**By- Abhilash Tyagi 21104019**

**Ojasvi Pandey 21104027**

class TicTacToe():

def \_\_init\_\_(self, player\_symbol):

# initialize the list of symbols

self.symbol\_list = []

# defines all nine symbols; all start off as blank

for i in range(9):

self.symbol\_list.append(" ")

# initializes the player symbol

self.player\_symbol = player\_symbol

def restart(self):

# clears the grid

for i in range(9):

self.symbol\_list[i] = " "

def draw\_grid(self):

# display the column headers

print("\n A B C\n")

# display first row

row\_one = " 1 " + self.symbol\_list[0]

row\_one += " ║ " + self.symbol\_list[1]

row\_one += " ║ " + self.symbol\_list[2]

print(row\_one)

# display divider

print(" ═══╬═══╬═══")

# display second row

row\_two = " 2 " + self.symbol\_list[3]

row\_two += " ║ " + self.symbol\_list[4]

row\_two += " ║ " + self.symbol\_list[5]

print(row\_two)

# display divider

print(" ═══╬═══╬═══")

# display third and last row

row\_three = " 3 " + self.symbol\_list[6]

row\_three += " ║ " + self.symbol\_list[7]

row\_three += " ║ " + self.symbol\_list[8]

print(row\_three, "\n")

def edit\_square(self, grid\_coord):

# swamps coordinates such as "1A" to "A1"

if grid\_coord[0].isdigit():

grid\_coord = grid\_coord[1] + grid\_coord[0]

# divides the coordinate

col = grid\_coord[0].capitalize()

row = grid\_coord[1]

# converts "A1" to 0, "C3" to 8, and so forth

grid\_index = 0

if row == "1":

if col == "A":

grid\_index = 0

elif col == "B":

grid\_index = 1

elif col == "C":

grid\_index = 2

elif row == "2":

if col == "A":

grid\_index = 3

elif col == "B":

grid\_index = 4

elif col == "C":

grid\_index = 5

elif row == "3":

if col == "A":

grid\_index = 6

elif col == "B":

grid\_index = 7

elif col == "C":

grid\_index = 8

if self.symbol\_list[grid\_index] == " ":

self.symbol\_list[grid\_index] = self.player\_symbol

def update\_symbol\_list(self, new\_symbol\_list):

for i in range(9):

self.symbol\_list[i] = new\_symbol\_list[i]

def did\_win(self, player\_symbol):

# local variable to replace unweildy self.symbol\_list

g = []

for i in range(9):

g.append(self.symbol\_list[i])

# likewise to replace self.player\_symbol

sym = player\_symbol

# check top row

if g[0] == sym and g[1] == sym and g[2] == sym:

return True

# check middle row

elif g[3] == sym and g[4] == sym and g[5] == sym:

return True

# check bottom row

elif g[6] == sym and g[7] == sym and g[8] == sym:

return True

# check left column

elif g[0] == sym and g[3] == sym and g[6] == sym:

return True

# check middle column

elif g[1] == sym and g[4] == sym and g[7] == sym:

return True

# check right column

elif g[2] == sym and g[5] == sym and g[8] == sym:

return True

# check top-right to bottom-left

elif g[2] == sym and g[4] == sym and g[6] == sym:

return True

# check top-left to bottom-right

elif g[0] == sym and g[4] == sym and g[8] == sym:

return True

# didn't win... yet!

return False

def is\_draw(self):

# see if all the spaces are used up

num\_blanks = 0

for i in range(9):

if self.symbol\_list[i] == " ":

num\_blanks += 1

# if the player didn't win and no spaces are left, it's a draw

if self.did\_win(self.player\_symbol) == False and num\_blanks == 0:

return True

else:

return False

Client

import socket # for networking

import pickle # for sending/receiving objects

# import the game

from tic\_tac\_toe import TicTacToe

HOST = '127.0.0.1' # the server's IP address

PORT = 12783 # the port we're connecting to

# connect to the host

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.connect((HOST, PORT))

print(f"\nConnected to {s.getsockname()}!")

# set up the game

player\_o = TicTacToe("O")

# allow the player to suggest playing again

rematch = True

while rematch == True:

# a header for an intense tic-tac-toe match!

print(f"\n\n T I C - T A C - T O E ")

# draw the grid

player\_o.draw\_grid()

# host goes first, client receives first

print(f"\nWaiting for other player...")

x\_symbol\_list = s.recv(1024)

x\_symbol\_list = pickle.loads(x\_symbol\_list)

player\_o.update\_symbol\_list(x\_symbol\_list)

# the rest is in a loop; if either player has won, it exits

while player\_o.did\_win("O") == False and player\_o.did\_win("X") == False and player\_o.is\_draw() == False:

# draw grid, ask for coordinate

print(f"\n Your turn!")

player\_o.draw\_grid()

player\_coord = input(f"Enter coordinate: ")

player\_o.edit\_square(player\_coord)

# draw grid again

player\_o.draw\_grid()

# pickle the symbol list and send it

o\_symbol\_list = pickle.dumps(player\_o.symbol\_list)

s.send(o\_symbol\_list)

# if the player won with the last move or it's a draw, exit the loop

if player\_o.did\_win("O") == True or player\_o.is\_draw() == True:

break

# wait to receive the symbol list and update it

print(f"\nWaiting for other player...")

x\_symbol\_list = s.recv(1024)

x\_symbol\_list = pickle.loads(x\_symbol\_list)

player\_o.update\_symbol\_list(x\_symbol\_list)

# end game messages

if player\_o.did\_win("O") == True:

print(f"Congrats, you won!")

elif player\_o.is\_draw() == True:

print(f"It's a draw!")

else:

print(f"Sorry, the host won.")

# host is being asked for a rematch, awaiting response

print(f"\nWaiting for host...")

host\_response = s.recv(1024)

host\_response = pickle.loads(host\_response)

client\_response = "N"

# if the host wants a rematch, then the client is asked

if host\_response == "Y":

print(f"\nThe host would like a rematch!")

client\_response = input("Rematch? (Y/N): ")

client\_response = client\_response.capitalize()

temp\_client\_resp = client\_response

# let the host know what the client decided

client\_response = pickle.dumps(client\_response)

s.send(client\_response)

# if the client wants a rematch, restart the game

if temp\_client\_resp == "Y":

player\_o.restart()

# if the client said no, then no rematch

else:

rematch = False

# if the host said no, then no rematch

else:

print(f"\nThe host does not want a rematch.")

rematch = False

spacer = input(f"\nThank you for playing!\nPress enter to quit...\n")

s.close()

Host

import socket

import pickle

from tic\_tac\_toe import TicTacToe

# Server configuration

HOST = '127.0.0.1'

PORT = 12783

# Set up the server

s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

s.bind((HOST, PORT))

s.listen(5)

try:

print(f"Server listening on {HOST}:{PORT}")

# Accept a connection from the client

client\_socket, client\_address = s.accept()

print(f"\nConnected to {client\_address}!")

# Set up the game

player\_x = TicTacToe("X")

# Allow the player to suggest playing again

rematch = True

while rematch:

# Header for an intense tic-tac-toe match!

print(f"\n\n T I C - T A C - T O E ")

# Loop until either player has won or it's a draw

while not player\_x.did\_win("X") and not player\_x.did\_win("O") and not player\_x.is\_draw():

# Draw the grid, ask for coordinate

print(f"\n Your turn!")

player\_x.draw\_grid()

player\_coord = input(f"Enter coordinate: ")

player\_x.edit\_square(player\_coord)

# Draw the grid again

player\_x.draw\_grid()

# Pickle the symbol list and send it

x\_symbol\_list = pickle.dumps(player\_x.symbol\_list)

client\_socket.send(x\_symbol\_list)

# If the player won with the last move or it's a draw, exit the loop

if player\_x.did\_win("X") or player\_x.is\_draw():

break

# Wait to receive the symbol list and update it

print(f"\nWaiting for other player...")

o\_symbol\_list = client\_socket.recv(1024)

o\_symbol\_list = pickle.loads(o\_symbol\_list)

player\_x.update\_symbol\_list(o\_symbol\_list)

# End game messages

if player\_x.did\_win("X"):

print(f"Congrats, you won!")

elif player\_x.is\_draw():

print(f"It's a draw!")

else:

print(f"Sorry, the client won.")

# Ask for a rematch

host\_response = input(f"\nRematch? (Y/N): ").capitalize()

# Pickle response and send it to the client

host\_response\_pickle = pickle.dumps(host\_response)

client\_socket.send(host\_response\_pickle)

# If the host doesn't want a rematch, we're done here

if host\_response == "N":

rematch = False

# If the host wants a rematch, receive the client's response

else:

print(f"Waiting for client response...")

client\_response = client\_socket.recv(1024)

client\_response = pickle.loads(client\_response)

# If the client doesn't want a rematch, exit the loop

if client\_response == "N":

print(f"\nThe client does not want a rematch.")

rematch = False

# If both the host and client want a rematch, restart the game

else:

player\_x.restart()

finally:

# Close the client socket and the server socket in the end

client\_socket.close()

s.close()