TICTACTOE

1. Code I made:

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Definitions of the Game Board Setup

x\_mark = " X "

O\_mark = " O "

blank = " "

row0 = [blank, blank, blank]

row1 = [blank, blank, blank]

row2 = [blank, blank, blank]

gameBoard = [row0, row1, row2]

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Funtion to Print The TicTacToe Game Board

def printBoard():

rows = 0

while (rows < 3):

print(gameBoard[rows])

rows = rows + 1

print (" ")

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Funtion to Add a Players Move to the Game Board

def addMove(mark, row, col):

gameBoard[row][col] = mark

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

# Main Program Code is Below

#Winning Combinations

if((gameBoard[0][0] =="x\_mark") and (gameBoard[0][1] =="x\_mark") and (gameBoard[0][2] =="x\_mark"),

(gameBoard[1][0] =="x\_mark") and (gameBoard[1][1] =="x\_mark") and (gameBoard[1][2] =="x\_mark"),

(gameBoard[2][0] =="x\_mark") and (gameBoard[2][1] =="x\_mark") and (gameBoard[2][2] =="x\_mark"),

(gameBoard[0][0] =="x\_mark") and (gameBoard[1][1] =="x\_mark") and (gameBoard[2][2] =="x\_mark"),

(gameBoard[2][0] =="x\_mark") and (gameBoard[1][1] =="x\_mark") and (gameBoard[0][2] =="x\_mark"),

(gameBoard[0][0] =="x\_mark") and (gameBoard[1][0] =="x\_mark") and (gameBoard[2][0] =="x\_mark"),

(gameBoard[0][1] =="x\_mark") and (gameBoard[1][1] =="x\_mark") and (gameBoard[2][1] =="x\_mark"),

(gameBoard[0][2] =="x\_mark") and (gameBoard[1][2] =="x\_mark") and (gameBoard[2][2] =="x\_mark")):

winningCombinations = True

if((gameBoard[0][0] =="O\_mark") and (gameBoard[0][1] =="O\_mark") and (gameBoard[0][2] =="O\_mark"),

(gameBoard[1][0] =="O\_mark") and (gameBoard[1][1] =="O\_mark") and (gameBoard[1][2] =="O\_mark"),

(gameBoard[2][0] =="O\_mark") and (gameBoard[2][1] =="O\_mark") and (gameBoard[2][2] =="O\_mark"),

(gameBoard[0][0] =="O\_mark") and (gameBoard[1][1] =="O\_mark") and (gameBoard[2][2] =="O\_mark"),

(gameBoard[2][0] =="O\_mark") and (gameBoard[1][1] =="O\_mark") and (gameBoard[0][2] =="O\_mark"),

(gameBoard[0][0] =="O\_mark") and (gameBoard[1][0] =="O\_mark") and (gameBoard[2][0] =="O\_mark"),

(gameBoard[0][1] =="O\_mark") and (gameBoard[1][1] =="O\_mark") and (gameBoard[2][1] =="O\_mark"),

(gameBoard[0][2] =="O\_mark") and (gameBoard[1][2] =="O\_mark") and (gameBoard[2][2] =="O\_mark")):

winningCombinations = True

# print the starting board

printBoard()

rowMove = int(input("Player x make a move: row = "))

if (rowMove > 2):

print("Bad row number, Try again...")

rowMove = int(input("Player x make a move: row = "))

colMove = int(input("Player x make a move: col = "))

if (colMove > 2):

print("Bad col number, Try again...")

colMove = int(input("Player x make a move: col = "))

addMove(O\_mark, rowMove, colMove)

printBoard()

rowMove = int(input("Player O make a move: row = "))

if (rowMove > 2):

print("Bad row number, Try again...")

rowMove = int(input("Player O make a move: row = "))

colMove = int(input("Player O make a move: col = "))

if (colMove > 2):

print("Bad col number, Try again...")

colMove = int(input("Player O make a move: col = "))

addMove(O\_mark, rowMove, colMove)

printBoard()

rowMove = int(input("Player x make a move: row = "))

if (rowMove > 2):

print("Bad row number, Try again...")

rowMove = int(input("Player x make a move: row = "))

colMove = int(input("Player x make a move: col = "))

if (colMove > 2):

print("Bad col number, Try again...")

colMove = int(input("Player x make a move: col = "))

addMove(O\_mark, rowMove, colMove)

printBoard()

rowMove = int(input("Player O make a move: row = "))

if (rowMove > 2):

print("Bad row number, Try again...")

rowMove = int(input("Player O make a move: row = "))

colMove = int(input("Player O make a move: col = "))

if (colMove > 2):

print("Bad col number, Try again...")

colMove = int(input("Player O make a move: col = "))

addMove(O\_mark, rowMove, colMove)

printBoard()

rowMove = int(input("Player x make a move: row = "))

if (rowMove > 2):

print("Bad row number, Try again...")

rowMove = int(input("Player x make a move: row = "))

colMove = int(input("Player x make a move: col = "))

if (colMove > 2):

print("Bad col number, Try again...")

colMove = int(input("Player x make a move: col = "))

addMove(O\_mark, rowMove, colMove)

printBoard()

rowMove = int(input("Player O make a move: row = "))

if (rowMove > 2):

print("Bad row number, Try again...")

rowMove = int(input("Player O make a move: row = "))

colMove = int(input("Player O make a move: col = "))

if (colMove > 2):

print("Bad col number, Try again...")

colMove = int(input("Player O make a move: col = "))

addMove(O\_mark, rowMove, colMove)

printBoard()

rowMove = int(input("Player x make a move: row = "))

if (rowMove > 2):

print("Bad row number, Try again...")

rowMove = int(input("Player x make a move: row = "))

colMove = int(input("Player x make a move: col = "))

if (colMove > 2):

print("Bad col number, Try again...")

colMove = int(input("Player x make a move: col = "))

addMove(O\_mark, rowMove, colMove)

printBoard()

rowMove = int(input("Player O make a move: row = "))

if (rowMove > 2):

print("Bad row number, Try again...")

rowMove = int(input("Player O make a move: row = "))

colMove = int(input("Player O make a move: col = "))

if (colMove > 2):

print("Bad col number, Try again...")

colMove = int(input("Player O make a move: col = "))

addMove(O\_mark, rowMove, colMove)

printBoard()

rowMove = int(input("Player x make a move: row = "))

if (rowMove > 2):

print("Bad row number, Try again...")

rowMove = int(input("Player x make a move: row = "))

colMove = int(input("Player x make a move: col = "))

if (colMove > 2):

print("Bad col number, Try again...")

colMove = int(input("Player x make a move: col = "))

addMove(O\_mark, rowMove, colMove)

printBoard()

# END OF PROGRAM

1. Codes I found online :

|  |
| --- |
| import sys |
|  | import random |
|  |  |
|  | person = input('Enter your name: ') |
|  | print('Hello', person) |
|  |  |
|  | moveCounter = 0 |
|  | matchOver = False |
|  |  |
|  | userMoves = [] |
|  | myMoves = [] |
|  |  |
|  | field = [[1, 2, 3], [4, 5, 6], [7, 8, 9]] |
|  |  |
|  | winnerPos = [["1", "2", "3"], ["1", "4", "7"], |
|  | ["1", "5", "9"], ["4", "5", "6"], |
|  | ["7", "8", "9"], ["2", "5", "8"], |
|  | ["3", "6", "9"], ["3", "5", "7"]] |
|  |  |
|  | firstAdvantage = ["5", "1", "3", "7", "9"] |
|  |  |
|  | secondAdvantage = [["5", "1"], ["5", "3"], ["5", "7"], |
|  | ["5", "9"], ["1", "3"], ["3", "9"], |
|  | ["7", "9"], ["1", "7"]] |
|  |  |
|  | thirdAdvantage = [["1", "3", "5"], ["3", "5", "9"], ["7", "5", "9"], |
|  | ["1", "5", "7"], ["1", "3", "9"], ["1", "3", "7"], |
|  | ["7", "9", "3"], ["2", "4", "1"], ["2", "3", "6"], |
|  | ["6", "9", "8"], ["4", "7", "8"]] |
|  |  |
|  | moves = ['1', '2', '3', '4', '5', '6', '7', '8', '9'] |
|  |  |
|  |  |
|  | "If player gets three in a row, player wins" |
|  | def checkWin(pos1, pos2, pos3, player): |
|  | if ((pos1 == player) and (pos2 == player) and (pos3 == player)): |
|  | playerWins = True |
|  | catsGame = False |
|  | else: |
|  | playerWins = False |
|  | return playerWins |
|  |  |
|  |  |
|  | "Check all possible win scenarios for a player." |
|  | def checkWinner(player): |
|  | for i in range(3): |
|  | win = checkWin(field[i][0], field[i][1], field[i][2], player) |
|  | if win: |
|  | if player == 'o': |
|  | print("I win!") |
|  | else: |
|  | print ("You win!") |
|  | catsGame = False |
|  | sys.exit(0) |
|  | for i in range(3): |
|  | win = checkWin(field[0][i], field[1][i], field[2][i], player) |
|  | if win: |
|  | if player == 'o': |
|  | print("I win!") |
|  | else: |
|  | print ("You win!") |
|  | catsGame = False |
|  | sys.exit(0) |
|  | win = checkWin(field[0][0], field[1][1], field[2][2], player) |
|  | if win: |
|  | if player == 'o': |
|  | print("I win!") |
|  | else: |
|  | print ("You win!") |
|  | catsGame = False |
|  | sys.exit(0) |
|  |  |
|  | win = checkWin(field[0][2], field[1][1], field[2][0], player) |
|  | if win: |
|  | if player == 'o': |
|  | print("I win!") |
|  | else: |
|  | print ("You win!") |
|  | catsGame = False |
|  | sys.exit(0) |
|  |  |
|  |  |
|  | "Print the board." |
|  | def printField(): |
|  | counter = 0 |
|  |  |
|  | for i in field[:2]: |
|  | for j in i[:2]: |
|  | print(" ", j, " | ", end="") |
|  | print(" ", field[counter][2]) |
|  | counter += 1 |
|  | print ("------------------") |
|  | print (" ", field[2][0], " | ", field[2][1], " | ", field[2][2]) |
|  |  |
|  |  |
|  | counter = 0 |
|  |  |
|  |  |
|  | "Defines coordinates of each numbered position on board." |
|  | def switch(x): |
|  | return { |
|  | '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] |
|  | }[x] |
|  |  |
|  |  |
|  | "Fills a position with players move." |
|  | def populate(x, side): |
|  | nums = switch(x) |
|  | field[nums[0]][nums[1]] = side |
|  |  |
|  |  |
|  | "Anticipate win or advantage for each player." |
|  | def think(): |
|  | if moveCounter == 0: |
|  | answer = "5" |
|  | elif moveCounter == 1: |
|  | if userMoves[-1] == "5": |
|  | answer = "1" |
|  | else: |
|  | answer = "5" |
|  | else: |
|  | answer = anticipateWin() |
|  | if answer == "0": |
|  | answer = anticipateUserWin() |
|  | if answer == "0": |
|  | answer = anticipateAdvantage() |
|  | if answer == "0": |
|  | answer = anticipateUserAdvantage() |
|  | if answer == "0": |
|  | answer = random.choice(moves) |
|  | myMoves.append(answer) |
|  | return answer |
|  |  |
|  |  |
|  | "Checks win and advantage scenario lists to inform next move." |
|  | def anticipate(posList, whoMoves): |
|  | answer = "0" |
|  | for lis in posList: |
|  | commonEl = set(whoMoves) & set(lis) |
|  | if len(commonEl) > 1: |
|  | for el in lis: |
|  | if el not in commonEl: |
|  | if el in moves: |
|  | answer = el |
|  | break |
|  | break |
|  | return answer |
|  |  |
|  |  |
|  | "Check if next move can win game." |
|  | def anticipateWin(): |
|  | answer = anticipate(winnerPos, myMoves) |
|  | return answer |
|  |  |
|  |  |
|  | "Check if user's next move can win game." |
|  | def anticipateUserWin(): |
|  | answer = anticipate(winnerPos, userMoves) |
|  | return answer |
|  |  |
|  |  |
|  | "Check if user's next move will give user advantage." |
|  | def anticipateUserAdvantage(): |
|  | if len(userMoves) < 2: |
|  | answer = anticipate(secondAdvantage, userMoves) |
|  | else: |
|  | answer = anticipate(thirdAdvantage, userMoves) |
|  | return answer |
|  |  |
|  |  |
|  | "Check if next move can give advantage." |
|  | def anticipateAdvantage(): |
|  | answer = "0" |
|  | if len(myMoves) < 2: |
|  | answer = anticipate(secondAdvantage, myMoves) |
|  | else: |
|  | answer = anticipate(thirdAdvantage, myMoves) |
|  | return answer |
|  |  |
|  |  |
|  | printField() |
|  |  |
|  |  |
|  | "User input decides who moves first." |
|  | moveFirst = input("Who moves first... you or me?: ") |
|  |  |
|  | if (moveFirst == "you"): |
|  | print ("My turn: ") |
|  |  |
|  | if (len(moves) > 0): |
|  | myMove = think() |
|  |  |
|  | if (myMove in moves): |
|  | moves.remove(myMove) |
|  | else: |
|  | print ("not in list") |
|  |  |
|  | populate(myMove, 'o') |
|  |  |
|  | printField() |
|  |  |
|  | moveCounter += 1 |
|  |  |
|  | catsGame = True |
|  |  |
|  | "Main loop of game lasts less than 9 moves." |
|  | while (moveCounter < 9): |
|  |  |
|  | move = input("Choose your move: ") |
|  |  |
|  | userMoves.append(move) |
|  |  |
|  | "Remove move position from list of possible moves." |
|  | if (len(moves) >= 0): |
|  | moves.remove(move) |
|  | else: |
|  | break |
|  |  |
|  | populate(move, 'x') |
|  |  |
|  | printField() |
|  |  |
|  | moveCounter += 1 |
|  |  |
|  | checkWinner('x') |
|  |  |
|  | print ("My turn: ") |
|  |  |
|  | if len(moves) > 0: |
|  | myMove = think() |
|  |  |
|  | if (myMove in moves): |
|  | moves.remove(myMove) |
|  | else: |
|  | print ("not in list") |
|  | break |
|  |  |
|  | populate(myMove, 'o') |
|  |  |
|  | printField() |
|  |  |
|  | moveCounter += 1 |
|  |  |
|  | checkWinner('o') |
|  |  |
|  | if catsGame: |
|  | print ("cats game...") |
|  | else: |
|  | print("...")  **Another Code :** |

**import** random

**class** BadInputError**(Exception):**

**pass**

**class** LogicError**(Exception):**

**pass**

*#===========GAMEBOARDS===========#*

blankBoard **=** **{**

'UL' **:** ' '**,** 'UM' **:** ' '**,** 'UR' **:** ' '**,**

'CL' **:** ' '**,** 'CM' **:** ' '**,** 'CR' **:** ' '**,**

'BL' **:** ' '**,** 'BM' **:** ' '**,** 'BR' **:** ' '**,**

**}**

debugBoard **=** **{**

'UL' **:** ' '**,** 'UM' **:** ' '**,** 'UR' **:** ' '**,**

'CL' **:** ' '**,** 'CM' **:** ' '**,** 'CR' **:** ' '**,**

'BL' **:** ' '**,** 'BM' **:** ' '**,** 'BR' **:** ' '**,**

**}**

invertedSpaces **=** **{**

'LU' **:** 'UL'**,** 'MU' **:** 'UM'**,** 'RU' **:** 'UR'**,**

'LC' **:** 'CL'**,** 'MC' **:** 'CM'**,** 'RC' **:** 'CR'**,**

'LB' **:** 'BL'**,** 'MB' **:** 'BM'**,** 'RB' **:** 'BR'**,**

**}**

*#===========DEFINITIONS===========#*

*'''Spaces'''*

spaces **=** **(**'UL'**,**'UM'**,**'UR'**,**'CL'**,**'CM'**,**'CR'**,**'BL'**,**'BM'**,**'BR'**)**

*'''Wins'''*

oWin **=** **(**'O'**,**'O'**,**'O'**)**

xWin **=** **(**'X'**,**'X'**,**'X'**)**

*'''Doubles'''*

oDoubles **=** **[(**' '**,**'O'**,**'O'**),(**'O'**,**' '**,**'O'**),(**'O'**,**'O'**,**' '**)]**

xDoubles **=** **[(**' '**,**'X'**,**'X'**),(**'X'**,**' '**,**'X'**),(**'X'**,**'X'**,**' '**)]**

*'''Input'''*

possibleInput **=** **[**key **for** key **in** blankBoard**]**

**for** key **in** invertedSpaces**:**

possibleInput**.**append**(**key**)**

*'''Space Types'''*

corners **=** **(**'UL'**,**'UR'**,**'BL'**,**'BR'**)**

sides **=** **(**'CL'**,**'CR'**,** 'UM'**,** 'BM'**)**

*'''Space Inversions'''*

horizontalFlip **=** **{**

'UL' **:** 'UR'**,**'UR' **:** 'UL'**,**

'CL' **:** 'CR'**,**'CR' **:** 'CL'**,**

'BL' **:** 'BR'**,**'BR' **:** 'BL'**,**

**}**

verticalFlip **=** **{**

'UL' **:** 'BL'**,** 'UM' **:** 'BM'**,** 'UR' **:** 'BR'**,**

'BL' **:** 'UL'**,** 'BM' **:** 'UM'**,** 'BR' **:** 'UR'**,**

**}**

*#===========OBJECTS===========#*

**class** ticBoard**():**

**def** \_\_init\_\_**(**self**,** mode**=**'blank'**,** copyBoard**=**None**):**

**if** mode **==** 'blank'**:**

self**.**board **=** **{**space**:**blankBoard**[**space**]** **for** space **in** blankBoard**}**

**elif** mode **==** 'debug'**:**

self**.**board **=** **{**space**:**debugBoard**[**space**]** **for** space **in** debugBoard**}**

**elif** mode **==** 'copy' **and** copyBoard **!=** None**:**

self**.**board **=** **{**space**:**copyBoard**.**board**[**space**]** **for** space **in** copyBoard**.**board**}**

**def** draw**(**self**):**

*'''Draw board'''*

**print()**

**print(**' L M R '**)**

**print(**'U: {} | {} | {} '**.**format**(**self**.**board**[**'UL'**],** self**.**board**[**'UM'**],** self**.**board**[**'UR'**]))**

**print(**' -----------'**)**

**print(**'C: {} | {} | {} '**.**format**(**self**.**board**[**'CL'**],** self**.**board**[**'CM'**],** self**.**board**[**'CR'**]))**

**print(**' -----------'**)**

**print(**'B: {} | {} | {} '**.**format**(**self**.**board**[**'BL'**],** self**.**board**[**'BM'**],** self**.**board**[**'BR'**]))**

**print()**

**def** place**(**self**,** symbol**,** space**):**

*'''Places a symbol at the designated space.'''*

**try:**

self**.**board**[**space**]** **=** symbol

**except:**

**raise** BadInputError**(**"{} is not a valid space for {}."**.**format**(**space**,** symbol**))**

**def** clear**(**self**):**

*'''Clears board of all symbols.'''*

self**.**board **=** **{**space**:**' ' **for** space **in** self**.**board**}**

**def** fieldReport**(**self**):**

*'''Returns dictionary of triads.'''*

report **=** **{}**

report**[(**'UL'**,**'UM'**,**'UR'**)]** **=** **(**self**.**board**[**'UL'**],**self**.**board**[**'UM'**],**self**.**board**[**'UR'**])**

report**[(**'CL'**,**'CM'**,**'CR'**)]** **=** **(**self**.**board**[**'CL'**],**self**.**board**[**'CM'**],**self**.**board**[**'CR'**])**

report**[(**'BL'**,**'BM'**,**'BR'**)]** **=** **(**self**.**board**[**'BL'**],**self**.**board**[**'BM'**],**self**.**board**[**'BR'**])**

report**[(**'UL'**,**'CL'**,**'BL'**)]** **=** **(**self**.**board**[**'UL'**],**self**.**board**[**'CL'**],**self**.**board**[**'BL'**])**

report**[(**'UM'**,**'CM'**,**'BM'**)]** **=** **(**self**.**board**[**'UM'**],**self**.**board**[**'CM'**],**self**.**board**[**'BM'**])**

report**[(**'UR'**,**'CR'**,**'BR'**)]** **=** **(**self**.**board**[**'UR'**],**self**.**board**[**'CR'**],**self**.**board**[**'BR'**])**

report**[(**'UL'**,**'CM'**,**'BR'**)]** **=** **(**self**.**board**[**'UL'**],**self**.**board**[**'CM'**],**self**.**board**[**'BR'**])**

report**[(**'UR'**,**'CM'**,**'BL'**)]** **=** **(**self**.**board**[**'UR'**],**self**.**board**[**'CM'**],**self**.**board**[**'BL'**])**

**return** report

**def** returnDoubles**(**self**,** report**):**

*'''Filters out report to only include triads close to winning. ie "[X,X, ]" or '[O, ,O]"'''*

doubles **=** **{}**

**for** triad **in** report**:**

**if** report**[**triad**]** **in** oDoubles **or** report**[**triad**]** **in** xDoubles**:**

doubles**[**triad**]** **=** report**[**triad**]**

**return** doubles

**def** checkWin**(**self**):**

*'''Returns True if there are three symbols in a row. False if otherwise.'''*

report **=** self**.**fieldReport**()**

**for** triad **in** report**:**

**if** report**[**triad**]** **==** oWin **or** report**[**triad**]** **==** xWin**:**

**return** True

**return** False

**def** checkEntry**(**self**,** entry**,** selected**):**

*'''Returns the entry and whether or not it is valid.'''*

entry **=** entry**.**upper**()**

**if** entry **in** invertedSpaces**:**

entry **=** invertedSpaces**[**entry**]**

**if** entry **not** **in** possibleInput**:**

**return** **{**'valid'**:**False**,**'entry'**:**entry**,** 'message'**:**'\n{} is not a valid entry!'**}**

**if** entry **not** **in** selected**:**

**return** **{**'valid'**:**True**,**'entry'**:**entry**}**

**else:**

**return** **{**'valid'**:**False**,**'entry'**:**entry**,** 'message'**:**'\n{} has already been selected!'**}**

**def** buildString**(**self**,** string**):**

**if** len**(**string**)** **!=** **9:**

**print(**'String is not correct length. Reformatting will occur.'**)**

string **=** string**[:9]**

**while** len**(**string**)** **<** **9:**

string **+=** '0'

**for** i **in** range**(9):**

**if** string**[**i**]** **==** '0'**:**

self**.**board**[**spaces**[**i**]]** **=** ' '

**elif** string**[**i**]** **==** '1'**:**

self**.**board**[**spaces**[**i**]]** **=** 'O'

**elif** string**[**i**]** **==** '2'**:**

self**.**board**[**spaces**[**i**]]** **=** 'X'

**def** blankSpaces**(**self**):**

*'''Returns list of free spaces remianing.'''*

**return** **[**space **for** space **in** self**.**board **if** self**.**board**[**space**]** **==** ' '**]**

**class** player**():**

**def** \_\_init\_\_**(**self**,** identity**):**

self**.**id **=** identity

self**.**score **=** **0**

self**.**match **=** **0**

self**.**symbol **=** ''

**def** setName**(**self**,** name**):**

*'''Define player's name.'''*

**if** **0** **<** len**(**str**(**name**))** **<** **20:**

self**.**name **=** name**.**title**()**

**return** False

**else:**

**return** True

**def** setSymbol**(**self**,** symbol**):**

**if** symbol**.**upper**()** **in** **[**'X'**,**'O'**]:**

self**.**symbol **=** symbol**.**upper**()**

**return** False

**else:**

**return** True

**def** win**(**self**):**

self**.**score **+=** **1**

**def** matchWin**(**self**):**

self**.**match **+=** **1**

**def** resetMatch**(**self**):**

self**.**match **=** **0**

**def** getSymbol**(**self**):**

**return** self**.**symbol

**def** getName**(**self**):**

**return** self**.**name

**def** getIdentity**(**self**):**

**return** self**.**id

**def** getScore**(**self**):**

**return** self**.**score

**def** getMatches**(**self**):**

**return** self**.**match

**class** computer**(**player**):**

**def** \_\_init\_\_**(**self**,** difficulty**=**'E'**):**

self**.**id **=** 'comp'

self**.**difficulty **=** difficulty**[0]**

self**.**setName**(**'Computer'**)**

self**.**setSymbol**(**'X'**)**

self**.**score **=** **0**

self**.**match **=** **0**

self**.**strategy **=** ''

self**.**tactic **=** ''

self**.**lastMove **=** ''

self**.**reiterate **=** False

**def** mapCoordinates**(**self**,** triad**):**

*'''Converts a entry from a triad tuple to a dictionary of*

*symbol : coordinate values.'''*

mapped **=** **{}**

coor **=** **0**

**for** coordinate **in** triad**[0]:**

mapped**[**coordinate**]** **=** triad**[1][**coor**]**

coor**+=1**

**return** mapped

**def** analyzeMap**(**self**,** mappedCoordinates**):**

*'''Returns empty value from a mapped coordinates dictionary.'''*

**for** key **in** mappedCoordinates**:**

**if** mappedCoordinates**[**key**]** **==** ' '**:**

**return** key

**def** defineStrategy**(**self**,** strategy**):**

*'''Play offensively (first turn) or defensively.'''*

**if** strategy **in** **[**'offensive'**,**'defensive'**]:**

self**.**strategy **=** strategy

**def** decideTactic**(**self**,** board**):**

*'''Decide tactic based on the first move or by making first move.'''*

**if** self**.**strategy **==** 'offensive'**:**

firstMove **=** random**.**choice**([**'center'**,**'corner'**])**

*#firstMove = 'corner'*

self**.**tactic **=** firstMove

**elif** self**.**strategy **==** 'defensive'**:**

**for** space **in** board**.**board**:**

**if** board**.**board**[**space**]** **==** 'O'**:**

**if** space **in** corners**:**

self**.**tactic **=** 'corner'

**elif** space **==** 'CM'**:**

self**.**tactic **=** 'center'

**else:**

self**.**tactic **=** 'side'

**def** clearStrategy**(**self**):**

self**.**strategy **=** ''

self**.**tactic **=** ''

**def** counter**(**self**,** doubles**):**

*'''Either place winning piece or stop opponent from winning.'''*

**if** doubles **!=** **{}:**

triad **=** doubles**.**popitem**()**

entry **=** self**.**analyzeMap**(**self**.**mapCoordinates**(**triad**))**

debug**(**d**,**'Countering'**)**

**return** **{**'counter'**:**True**,** 'entry'**:**entry**}**

**return** **{**'counter'**:**False**,** 'entry'**:**''**}**

**def** trapSimulation**(**self**,** board**,** report**,** pool**):**

*'''Simulate different moves to trap opponent.'''*

**for** coordinate **in** pool**:**

simulatedBoard **=** ticBoard**(**'copy'**,**board**)**

simulatedBoard**.**place**(**self**.**getSymbol**(),**coordinate**)**

simulatedDoubles **=** board**.**returnDoubles**(**simulatedBoard**.**fieldReport**())**

soDoubles **=** **{**key**:**simulatedDoubles**[**key**]** **for** key **in** simulatedDoubles **if** 'O' **in** simulatedDoubles**[**key**]}**

sxDoubles **=** **{**key**:**simulatedDoubles**[**key**]** **for** key **in** simulatedDoubles **if** 'X' **in** simulatedDoubles**[**key**]}**

**if** len**(**soDoubles**)** **==** **0** **and** len**(**sxDoubles**)** **>** **1:**

debug**(**d**,**'Trapping'**)**

**return** **{**'trap'**:**True**,** 'entry'**:**coordinate**}**

**return** **{**'trap'**:**False**,** 'entry'**:**coordinate**}**

**def** offensiveStrategy**(**self**,** board**):**

*'''Provide offensive move based on a certain tactic.'''*

**if** self**.**tactic **==** 'center'**:**

**if** len**(**board**.**blankSpaces**())** **==** **9:**

debug**(**d**,**'Begin Center'**)**

**return** **{**'offensive'**:**True**,** 'entry'**:**'CM'**}**

**elif** len**(**board**.**blankSpaces**())** **==** **7:**

**for** corner **in** corners**:**

**if** board**.**board**[**corner**]** **==** 'O' **and** board**.**board**[**verticalFlip**[**horizontalFlip**[**corner**]]]** **==** ' '**:**

debug**(**d**,**'Countering Corner'**)**

**return** **{**'offensive'**:**True**,** 'entry'**:** verticalFlip**[**horizontalFlip**[**corner**]]}**

**return** **{**'offensive'**:**False**,** 'entry'**:**''**}**

**else:**

**return** **{**'offensive'**:**False**,** 'entry'**:**''**}**

**elif** self**.**tactic **==** 'corner'**:**

**if** len**(**board**.**blankSpaces**())** **==** **9:**

debug**(**d**,**'Begin Corner'**)**

**return** **{**'offensive'**:**True**,** 'entry'**:**random**.**choice**(**corners**)}**

**else:**

**if** board**.**board**[**'CM'**]** **!=** 'O'**:**

**if** self**.**lastMove **!=** ''**:**

**if** board**.**board**[**horizontalFlip**[**self**.**lastMove**]]** **==** ' ' **and** board**.**board**[**self**.**lastMove**[0]** **+** 'M'**]** **!=** 'O'**:**

debug**(**d**,**'Horizontal Flip'**)**

**return** **{**'offensive'**:**True**,** 'entry'**:**horizontalFlip**[**self**.**lastMove**]}**

**elif** board**.**board**[**horizontalFlip**[**self**.**lastMove**]]** **==** 'O'**:**

debug**(**d**,**'Invert'**)**

**return** **{**'offensive'**:**True**,** 'entry'**:**verticalFlip**[**horizontalFlip**[**self**.**lastMove**]]}**

**else:**

debug**(**d**,**'Vertical Flip'**)**

**return** **{**'offensive'**:**True**,** 'entry'**:**verticalFlip**[**self**.**lastMove**]}**

**if** board**.**board**[**'CM'**]** **==** 'O'**:**

**for** space **in** board**.**board**:**

**if** board**.**board**[**space**]** **==** 'X' **and** space **in** corners**:**

debug**(**d**,**'Form XOX'**)**

**return** **{**'offensive'**:**True**,** 'entry'**:**horizontalFlip**[**verticalFlip**[**space**]]}**

**else:**

**return** **{**'offensive'**:**False**,** 'entry'**:**''**}**

**def** defensiveStrategy**(**self**,** board**):**

*'''Provide defensive move based on a certain tactic.'''*

**if** self**.**tactic **==** 'center'**:** *#Keep Selecting Corners*

**for** corner **in** corners**:**

**if** board**.**board**[**corner**]** **==** ' '**:**

debug**(**d**,**'Get Corners'**)**

**return** **{**'defense'**:**True**,** 'entry'**:**corner**}**

**elif** self**.**tactic **==** 'corner'**:**

**if** board**.**board**[**'CM'**]** **==** ' '**:** *#Get Center*

debug**(**d**,**'Secure Center'**)**

**return** **{**'defense'**:**True**,** 'entry'**:**'CM'**}**

**else:**

**if** len**(**board**.**blankSpaces**())** **==** **6:**

cornersFound **=** **0**

**for** corner **in** corners**:**

**if** board**.**board**[**corner**]** **==** 'O'**:**

cornersFound **+=** **1**

**if** cornersFound **==** **2:**

**for** side **in** sides**:**

**if** board**.**board**[**side**]** **==** ' '**:**

debug**(**d**,**'Two Corners'**)**

**return** **{**'defense'**:**True**,** 'entry'**:**side**}**

**else:**

self**.**strategy **=** 'offensive'

self**.**tactic **=** 'center'

self**.**reiterate **=** True

debug**(**d**,**'Retrategizing'**)**

**return** **{**'defense'**:**False**,** 'entry'**:**''**}**

**elif** self**.**tactic **==** 'side'**:**

**if** board**.**board**[**'CM'**]** **==** ' '**:** *#Get Center*

**return** **{**'defense'**:**True**,** 'entry'**:**'CM'**}**

**else:**

**if** len**(**board**.**blankSpaces**())** **==** **6:**

report **=** board**.**fieldReport**()**

**for** triad **in** report**:**

**if** triad **==** **(**'O'**,**'X'**,**'O'**):**

debug**(**d**,**'OXO Kill'**)**

**return** **{**'defense'**:**True**,** 'entry'**:**random**.**choice**(**corner**)}**

**return** **{**'defense'**:**False**,** 'entry'**:**''**}**

**def** think**(**self**,** board**):**

*'''Return best possible move for a given situation.'''*

*### Query Board for Information ###*

**while** True**:**

report **=** board**.**fieldReport**()**

totalDoubles **=** board**.**returnDoubles**(**report**)**

oDoubles **=** **{**key**:**totalDoubles**[**key**]** **for** key **in** totalDoubles **if** 'O' **in** totalDoubles**[**key**]}**

xDoubles **=** **{**key**:**totalDoubles**[**key**]** **for** key **in** totalDoubles **if** 'X' **in** totalDoubles**[**key**]}**

pool **=** board**.**blankSpaces**()**

**if** pool **==** **[]:**

**return**

*### Check for Winning Counters ###*

counterMove **=** self**.**counter**(**xDoubles**)**

**if** counterMove**[**'counter'**]:**

self**.**lastMove **=** counterMove**[**'entry'**]**

**return** counterMove**[**'entry'**]**

*### Check for Losing Counters ###*

counterMove **=** self**.**counter**(**oDoubles**)**

**if** counterMove**[**'counter'**]:**

self**.**lastMove **=** counterMove**[**'entry'**]**

**return** counterMove**[**'entry'**]**

*### Check for Trapping Moves ###*

trapMove **=** self**.**trapSimulation**(**board**,** report**,** pool**)**

**if** trapMove**[**'trap'**]:**

self**.**lastMove **=** trapMove**[**'entry'**]**

**return** trapMove**[**'entry'**]**

*### Strategize ###*

**if** self**.**strategy **==** ''**:**

**if** len**(**board**.**blankSpaces**())** **==** **9:**

self**.**strategy **=** 'offensive'

**else:**

self**.**strategy **=** 'defensive'

**if** self**.**tactic **==** ''**:**

self**.**decideTactic**(**board**)**

**if** self**.**strategy **==** 'offensive'**:**

offenseMove **=** self**.**offensiveStrategy**(**board**)**

**if** offenseMove**[**'offensive'**]:**

self**.**lastMove **=** offenseMove**[**'entry'**]**

**return** offenseMove**[**'entry'**]**

**else:**

defenseMove **=** self**.**defensiveStrategy**(**board**)**

**if** defenseMove**[**'defense'**]:**

self**.**lastMove **=** defenseMove**[**'entry'**]**

**return** defenseMove**[**'entry'**]**

*### Random Guess ###*

**if** self**.**reiterate**:**

self**.**reiterate **=** False

**else:**

debug**(**d**,**'Random Entry'**)**

entry **=** random**.**choice**(**pool**)**

self**.**lastMove **=** entry

**return** entry

**class** debugger**():**

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

self**.**active **=** True

*#===========HELPER FUNCTIONS===========#*

**def** nextTurn**(**turnList**,** currentTurn**):**

currentIndex **=** turnList**.**index**(**currentTurn**)**

**if** **(**currentIndex **+** **1)** **==** len**(**turnList**):**

**return** turnList**[0]**

**else:**

**return** turnList**[**currentIndex**+1]**

**def** debug**(**debugObject**,**statement**):**

**if** debugObject**.**active**:**

**print(**statement**)**

*#===========GAME FUNCTIONS=============#*

d **=** debugger**()**

**def** TicTacToe**(**debugging**=**True**):**

**if** **not** debugging**:**

d**.**active **=** False

*###MENUS###*

**def** mainMenu**():**

difficulty **=** 'Easy'

players **=** **{}**

debugStatus **=** ''

**while** True**:**

**if** d**.**active**:**

debugStatus **=** 'Enabled'

**else:**

debugStatus **=** 'Disabled'

**print(**'\t\tTic-Tac-Toe'**)**

**print(**'\n\t1. One Player'**)**

**print(**'\t2. Two Players'**)**

**if** players **!=** **{}:**

**print(**'\t\tA. Rematch'**)**

**print(**'\n\t3. Computer Difficulty:'**,**difficulty**)**

**print(**'\t4. Debugging'**,**debugStatus**)**

selection **=** str**(**input**(**'\nSelect Game Mode: '**))**

**while** selection **not** **in** **[**'1'**,** '2'**,** '3'**,** '4'**,** 'A'**,** 'a'**,** 'escape'**]:**

**print(**'\nSelection Invalid'**)**

selection **=** str**(**input**(**'\nSelect Game Mode: '**))**

**if** selection **==** '1'**:**

**print()**

players **=** singlePlayer**(**difficulty**)**

**print()**

players **=** gameplay**(**players**)**

**elif** selection **==** '2'**:**

**print()**

players **=** multiPlayer**()**

**print()**

players **=** gameplay**(**players**)**

**elif** selection **==** '3'**:**

**print()**

**if** difficulty **==** 'Easy'**:**

difficulty **=** 'Medium'

**elif** difficulty **==** 'Medium'**:**

difficulty **=** 'Hard'

**elif** difficulty **==** 'Hard'**:**

difficulty **=** 'Impossible'

**else:**

difficulty **=** 'Easy'

**elif** selection **==** '4'**:**

**print()**

**if** d**.**active**:**

d**.**active **=** False

**else:**

d**.**active **=** True

**elif** selection **in** **[**'A'**,**'a'**]:**

**if** players **!=** **{}:**

**print()**

players **=** gameplay**(**players**)**

**else:**

**print(**'Not an Option'**)**

**elif** selection **==** 'escape'**:**

**break**

**else:**

**raise** BadInputError**(**'Data Provided Has No Function'**)**

**def** singlePlayer**(**difficulty**):**

*'''Returns dictionary of players for singleplayer gameplay.'''*

players **=** **{}**

newPlayer **=** player**(**'play1'**)**

**print(**'Player 1'**,**end**=**' '**)**

**if** **not** d**.**active**:**

nameEntry **=** str**(**input**(**'please enter your name: '**))**

**while** newPlayer**.**setName**(**nameEntry**):**

**print(**'Invalid Entry!'**)**

**print(**'Player 1'**,**end**=**' '**)**

nameEntry **=** str**(**input**(**'please enter your name: '**))**

**else:**

newPlayer**.**setName**(**"Debug"**)**

newPlayer**.**setSymbol**(**'O'**)**

players**[**'play1'**]** **=** newPlayer

players**[**'comp'**]** **=** computer**(**difficulty**)**

**return** players

**def** multiPlayer**():**

*'''Returns dictionary of players for multiplayer gameplay.'''*

symbols **=** **[**'X'**,**'O'**]**

players **=** **{}**

**for** identity **in** **[**'play1'**,**'play2'**]:**

**if** identity **==** 'play1'**:**

title **=** 'Player 1'

**else:**

title **=** 'Player 2'

newPlayer **=** player**(**identity**)**

**print(**title**,**end**=**' '**)**

nameEntry **=** str**(**input**(**'please enter your name: '**))**

**while** newPlayer**.**setName**(**nameEntry**):**

**print(**'Invalid Entry!'**)**

**print(**title**,**end**=**' '**)**

nameEntry **=** str**(**input**(**'please enter your name: '**))**

**if** identity **==** 'play1'**:**

symbolEntry **=** str**(**input**(**'O or X: '**))**

**while** newPlayer**.**setSymbol**(**symbolEntry**):**

**print(**'Invalid Entry!'**)**

symbolEntry **=** str**(**input**(**'O or X: '**))**

symbols**.**remove**(**symbolEntry**.**upper**())**

**else:**

newPlayer**.**setSymbol**(**symbols**[0])**

players**[**identity**]** **=** newPlayer

**return** players

**def** gameplay**(**players**):**

*'''Provides turn system for a Tic Tac Toe Game.'''*

**if** 'comp' **not** **in** players**:**

**print(**'Beginning Game, {} vs {}'**.**format**(**players**[**'play1'**].**getName**(),**players**[**'play2'**].**getName**()))**

**else:**

**print(**'Beginning Game, {} vs the Computer'**.**format**(**players**[**'play1'**].**getName**()))**

**print(**"Win Two Matches In a Row to Be Victorious"**)**

board **=** ticBoard**(**mode**=**'blank'**)**

turnList **=** list**(**players**.**keys**())**

firstTurn **=** random**.**choice**(**turnList**)**

*#firstTurn = 'comp'*

turn **=** firstTurn

selected **=** **[]**

**while** True**:**

board**.**draw**()**

**if** turn **!=** 'comp'**:**

**print(**players**[**turn**].**getName**(),**'please select a space.'**)**

selection **=** str**(**input**(**'Space: '**)).**upper**()**

**if** **not** d**.**active **or** selection **not** **in** **[**"RESET"**,** "END"**]:**

errorCheck **=** board**.**checkEntry**(**selection**,**selected**)**

**while** **not** errorCheck**[**'valid'**]:**

errorMessage **=** errorCheck**[**'message'**].**format**(**selection**)**

**print(**errorMessage**)**

**print(**players**[**turn**].**getName**(),**'please select a space.'**)**

selection **=** str**(**input**(**'Space: '**)).**upper**()**

errorCheck **=** board**.**checkEntry**(**selection**,**selected**)**

**if** selection **==** 'END'**:**

**break**

**else:**

**print(**'Computer Turn'**)**

selection **=** players**[**'comp'**].**think**(**board**)**

errorCheck **=** board**.**checkEntry**(**selection**,**selected**)**

**print(**'Computer Chooses {}'**.**format**(**selection**))**

**if** selection **!=** 'RESET' **or** **not** d**.**active**:**

board**.**place**(**players**[**turn**].**getSymbol**(),** errorCheck**[**'entry'**])**

selected**.**append**(**selection**)**

**if** board**.**checkWin**()** **and** selection **!=** 'RESET'**:**

board**.**draw**()**

selected **=** **[]**

winner **=** turn

loser **=** nextTurn**(**turnList**,** turn**)**

**if** players**[**winner**].**getMatches**()** **==** **1:**

**print(**players**[**turn**].**getName**(),**end**=**' '**)**

str**(**input**(**'wins!'**))**

players**[**turn**].**win**()**

players**[**loser**].**resetMatch**()**

players**[**winner**].**resetMatch**()**

**break** *#END GAME*

**elif** players**[**winner**].**getMatches**()** **==** **0:**

**print(**players**[**turn**].**getName**(),**end**=**' '**)**

players**[**winner**].**matchWin**()**

players**[**loser**].**resetMatch**()**

str**(**input**(**'won a match! Beginning next round.'**))**

**if** 'comp' **in** players**:**

players**[**'comp'**].**clearStrategy**()**

board**.**clear**()**

turn **=** loser

firstTurn **=** loser

**else:**

**if** len**(**board**.**blankSpaces**())** **==** **0** **or** selection **==** 'RESET'**:**

board**.**draw**()**

str**(**input**(**'Draw! Beginning next round.'**))**

**if** 'comp' **in** players**:**

players**[**'comp'**].**clearStrategy**()**

players**[**turn**].**resetMatch**()**

players**[**nextTurn**(**turnList**,** turn**)].**resetMatch**()**

board**.**clear**()**

firstTurn **=** nextTurn**(**turnList**,** firstTurn**)**

turn **=** firstTurn

selected **=** **[]**

**else:**

turn **=** nextTurn**(**turnList**,** turn**)**

**try:**

**print()**

**print(**'\t\tCurrent Score\n'**)**

**print(**'\t'**+**players**[**winner**].**getName**()+**'\t\t\t'**+**str**(**players**[**winner**].**getScore**()))**

**print(**'\t'**+**players**[**loser**].**getName**()+**'\t\t\t'**+**str**(**players**[**loser**].**getScore**()))**

**print(**'\n==========================================\n'**)**

**except:**

**print(**'\tNo Scores to Show.\n'**)**

**return** players

mainMenu**()** *#Load Main Menu First*

activeDebug **=** False

TicTacToe**(**activeDebug**)** *#Begin Program*