horizontal line

Tic-Tac-Toe Game

**22th JAN 2024**

## OVERVIEW

⁠ The provided code is a Tic Tac Toe game implemented using Python's tkinter library. It offers two modes of play: single-player (against the computer) and multi-player (against another human player).

## GOALS

1. Working Tic-Tac-Toe game.
2. Project documentation

## ROLES

**Project Manager (PM):** **Abhinav Kumar**

**Responsibilities**:

* Oversee project progress.
* Coordinate team meetings.
* Ensure deadlines are met.

**Key Deliverables:**

* Project plan and timeline.
* Regular progress reports.

**Lead Developer (LD):** **Aakarshit Rathore**

**Responsibilities:**

* Write core game logic.
* Ensure code quality and efficiency.

**Key Deliverables:**

* Core game logic and algorithms.
* Code documentation.

**Interface Designer (ID):** **Omkar R Prasad**

**Responsibilities:**

* Design user interface for the game.
* Implement input/output.

**Key Deliverables:**

* User interface code.
* User experience report.

**Quality Assurance (QA):** **MD Talib**

**Responsibilities:**

* Test the game for bugs and issues.
* Document and report any issues to the LD.
* Suggest key improvements.

**Key Deliverables:**

* Test cases and results.
* Bug and improvement reports.

## PROJECT PHASES:

# Planning (Day 1):

* Brainstorming session
* Assign roles

**Development (Days 2-3):**

* LD and ID develops the game logic.
* The PM coordinates the work.

**Integration and Testing (Day 4)**

* QA tests the game.
* Bug fixing.

**Presentation and Review (Day 5).**

* Conduct a code review session.
* Reflect on learning and project experience

## FINAL DELIVERABLES:

* Working Tic-Tac-Toe game.
* Project documentation.
* Presentation and review materials.

## CODE:

# Tic Tac Toe game with GUI

# using tkinter

# importing all necessary libraries

import random

import tkinter

from tkinter import \*

from functools import partial

from tkinter import messagebox

from copy import deepcopy

# sign variable to decide the turn of which player

sign = 0

# Creates an empty board

global board

board = [[" " for x in range(3)] for y in range(3)]

# Check l(O/X) won the match or not

# according to the rules of the game

def winner(b, l):

return ((b[0][0] == l and b[0][1] == l and b[0][2] == l) or

(b[1][0] == l and b[1][1] == l and b[1][2] == l) or

(b[2][0] == l and b[2][1] == l and b[2][2] == l) or

(b[0][0] == l and b[1][0] == l and b[2][0] == l) or

(b[0][1] == l and b[1][1] == l and b[2][1] == l) or

(b[0][2] == l and b[1][2] == l and b[2][2] == l) or

(b[0][0] == l and b[1][1] == l and b[2][2] == l) or

(b[0][2] == l and b[1][1] == l and b[2][0] == l))

# Configure text on button while playing with another player

def get\_text(i, j, gb, l1, l2):

global sign

if board[i][j] == ' ':

if sign % 2 == 0:

l1.config(state=DISABLED)

l2.config(state=ACTIVE)

board[i][j] = "X"

else:

l2.config(state=DISABLED)

l1.config(state=ACTIVE)

board[i][j] = "O"

sign += 1

button[i][j].config(text=board[i][j])

if winner(board, "X"):

gb.destroy()

box = messagebox.showinfo("Winner", "Player 1 won the match")

elif winner(board, "O"):

gb.destroy()

box = messagebox.showinfo("Winner", "Player 2 won the match")

elif(isfull()):

gb.destroy()

box = messagebox.showinfo("Tie Game", "Tie Game")

# Check if the player can push the button or not

def isfree(i, j):

return board[i][j] == " "

# Check the board is full or not

def isfull():

flag = True

for i in board:

if(i.count(' ') > 0):

flag = False

return flag

# Create the GUI of game board for play along with another player

def gameboard\_pl(game\_board, l1, l2):

global button

button = []

for i in range(3):

m = 3+i

button.append(i)

button[i] = []

for j in range(3):

n = j

button[i].append(j)

get\_t = partial(get\_text, i, j, game\_board, l1, l2)

button[i][j] = Button(

game\_board, bd=5, command=get\_t, height=4, width=8)

button[i][j].grid(row=m, column=n)

game\_board.mainloop()

# Decide the next move of system

def pc():

possiblemove = []

for i in range(len(board)):

for j in range(len(board[i])):

if board[i][j] == ' ':

possiblemove.append([i, j])

move = []

if possiblemove == []:

return

else:

for let in ['O', 'X']:

for i in possiblemove:

boardcopy = deepcopy(board)

boardcopy[i[0]][i[1]] = let

if winner(boardcopy, let):

return i

corner = []

for i in possiblemove:

if i in [[0, 0], [0, 2], [2, 0], [2, 2]]:

corner.append(i)

if len(corner) > 0:

move = random.randint(0, len(corner)-1)

return corner[move]

edge = []

for i in possiblemove:

if i in [[0, 1], [1, 0], [1, 2], [2, 1]]:

edge.append(i)

if len(edge) > 0:

move = random.randint(0, len(edge)-1)

return edge[move]

# Configure text on button while playing with system

def get\_text\_pc(i, j, gb, l1, l2):

global sign

if board[i][j] == ' ':

if sign % 2 == 0:

l1.config(state=DISABLED)

l2.config(state=ACTIVE)

board[i][j] = "X"

else:

button[i][j].config(state=ACTIVE)

l2.config(state=DISABLED)

l1.config(state=ACTIVE)

board[i][j] = "O"

sign += 1

button[i][j].config(text=board[i][j])

x = True

if winner(board, "X"):

gb.destroy()

x = False

box = messagebox.showinfo("Winner", "Player won the match")

elif winner(board, "O"):

gb.destroy()

x = False

box = messagebox.showinfo("Winner", "Computer won the match")

elif(isfull()):

gb.destroy()

x = False

box = messagebox.showinfo("Tie Game", "Tie Game")

if(x):

if sign % 2 != 0:

move = pc()

button[move[0]][move[1]].config(state=DISABLED)

get\_text\_pc(move[0], move[1], gb, l1, l2)

# Create the GUI of game board for play along with system

def gameboard\_pc(game\_board, l1, l2):

global button

button = []

for i in range(3):

m = 3+i

button.append(i)

button[i] = []

for j in range(3):

n = j

button[i].append(j)

get\_t = partial(get\_text\_pc, i, j, game\_board, l1, l2)

button[i][j] = Button(

game\_board, bd=5, command=get\_t, height=4, width=8)

button[i][j].grid(row=m, column=n)

game\_board.mainloop()

# Initialize the game board to play with system

def withpc(game\_board):

game\_board.destroy()

game\_board = Tk()

game\_board.title("Tic Tac Toe")

l1 = Button(game\_board, text="Player : X", width=10)

l1.grid(row=1, column=1)

l2 = Button(game\_board, text="Computer : O",

width=10, state=DISABLED)

l2.grid(row=2, column=1)

gameboard\_pc(game\_board, l1, l2)

# Initialize the game board to play with another player

def withplayer(game\_board):

game\_board.destroy()

game\_board = Tk()

game\_board.title("Tic Tac Toe")

l1 = Button(game\_board, text="Player 1 : X", width=10)

l1.grid(row=1, column=1)

l2 = Button(game\_board, text="Player 2 : O",

width=10, state=DISABLED)

l2.grid(row=2, column=1)

gameboard\_pl(game\_board, l1, l2)

# main function

def play():

menu = Tk()

menu.geometry("250x250")

menu.title("Tic Tac Toe")

wpc = partial(withpc, menu)

wpl = partial(withplayer, menu)

head = Button(menu, text="---Welcome to tic-tac-toe---",

activeforeground='red',

activebackground="yellow", bg="red",

fg="yellow", width=500, font='summer', bd=5)

B1 = Button(menu, text="Single Player", command=wpc,

activeforeground='red',

activebackground="yellow", bg="red",

fg="yellow", width=500, font='summer', bd=5)

B2 = Button(menu, text="Multi Player", command=wpl, activeforeground='red',

activebackground="yellow", bg="red", fg="yellow",

width=500, font='summer', bd=5)

B3 = Button(menu, text="Exit", command=menu.quit, activeforeground='red',

activebackground="yellow", bg="red", fg="yellow",

width=500, font='summer', bd=5)

head.pack(side='top')

B1.pack(side='top')

B2.pack(side='top')

B3.pack(side='top')

menu.mainloop()

# Call main function

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

play()

## OUTPUT

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**QUALITY ASSURANCE TEST REPORT**

There are no explicit bugs or vulnerabilities in this code. However, there are a few potential areas for improvement:

1. **User Input Validation**: The code does not validate user input, which could potentially lead to errors if the user enters invalid data. For instance, the `isfull()` function could be improved by checking if the user input is within the valid range (0-2 for both rows and columns).

2. **Code Readability and Structure**: The code could benefit from better organization and commenting. This would make it easier for other developers to understand and maintain the code in the future.

3. **Error Handling**: The code does not handle exceptions or errors that may occur during gameplay. For example, if a player tries to make a move on a cell that is already occupied, the code simply overwrites the existing value. It would be better to display an error message to the user in such cases.

4. **Computer Player Strategy:** The current strategy used by the computer player is quite basic. It randomly chooses a move from the available cells. This strategy can be improved by incorporating more advanced AI techniques, such as minimax or Monte Carlo tree search algorithms.

In conclusion, while there are no explicit bugs or vulnerabilities in the provided code, there is room for improvement in terms of user input validation, error handling, computer player strategy, and GUI design. ⁠

## FUTURE SCOPE

1. **GUI Improvements:** The graphical user interface (GUI) could be improved by using more modern and visually appealing design elements. Additionally, the GUI could be made more responsive and user-friendly by incorporating features such as undo/redo and the ability to reset the game board.
2. **BOT Model improvements:** The (PvC) Person vs computer system currently runs on a randomizer function where the computer selects a random solution out of the available solution of the computer. We will try to implement (RL) reinforcement learning into the system for a responsive solution to every case in the grid.
3. **Choice of the grid:** The current model consists of a singular grid format i.e. 3x3. We will try and implement a multiple grid option for the user e.g. 4x4, 5x5, etc. This is proposed to improve the UI/UX experience.