TIC-TAC-TOE
A MINI PROJECT REPORT
18CSC207J - ADVANCED PROGRAMMING PRACTICE

Submitted by

Tanuja Kharol[RA2111003011808]

A. Lakshmi Nikitha [RA2111003011810]

Under the guidance of

Dr. Sivakumar S

Assistant Professor, Department of Computer Science and Engineering

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE & ENGINEERING

of

FACULTY OF ENGINEERING AND TECHNOLOGY

S.R.M. Nagar, Kattankulathur, Chengalpattu District

MAY 2023

SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

(Under Section 3 of UGC Act, 1956)

BONAFIDE CERTIFICATE

Certified that Mini project report titled **"TIC TAC TOE"** is the bonafide work of **Tanuja Kharol (RA2111003011808) and A. Lakshmi Nikitha (RA2111003011810)** who carried out the minor project under my supervision. Certified further, that to the best of my knowledge, the work reported herein does not form any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

Dr. Sivakumar S
Assistant Professor
Department of Computing Technologies

SIGNATURE

Dr. M. Pushpalatha
HEAD OF THE DEPARTMENT
Professor & Head
Department of Computing Technologies

Table Content

ABSTRACT

The above project is a Python implementation of the popular Tic Tac Toe game using the Tkinter library for the graphical user interface. The game features a 3x3 grid where two players take turns marking X's and O's in the grid until one of the players wins by placing three marks in a row, column, or diagonal, or the game ends in a draw if no player wins. The game also includes a "Play Again" button that resets the game board and allows the players to play again. The game's functionality is implemented through various methods in a class called `TicTacToe`, which is initialized with the Tkinter window and creates the game board and buttons through its `create_board` method. The game's state and logic are maintained through instance variables and methods like `button_click`, `change_turn`, `check_win`, `check_draw`, and `show_message`. When the game is over, a pop-up window is displayed showing the winner or a draw message, along with a "Play Again" button that resets the game board and destroys the pop-up window.

INTRODUCTION

The Tic Tac Toe game is a popular and easy-to-learn game that has been enjoyed by people of all ages for generations. It is played on a 3x3 grid where two players take turns marking X's and O's until one of the players achieves three in a row, column, or diagonal or the game ends in a draw. In this project, we have implemented the game using Python's Tkinter library to create a graphical user interface, making it more engaging and interactive for users.

The game's state and logic are managed through a `TicTacToe` class, which initializes the game board and buttons, tracks player turns, checks for wins and draws, and displays messages when the game is over. The class also includes a reset button that allows players to play the game again after the game is over. Through this project, we aim to provide users with a fun and easy way to play the classic Tic Tac Toe game on their computers, while also demonstrating the capabilities of Python's Tkinter library for creating graphical user interfaces.

METHODOLOGY

The methodology for implementing the Tic Tac Toe game using Python's Tkinter library can be broken down into several steps:

1. Importing necessary modules: First, we import the Tkinter module to create the graphical user interface and random module for selecting which player goes first.
2. Defining the `TicTacToe` class: We create a `TicTacToe` class that initializes the game board, creates the game's buttons, and sets up the necessary instance variables to keep track of the game's state.
3. Creating the game board and buttons: Using the `TicTacToe` class, we create a 3x3 grid of buttons and associate each button with a callback function that gets called when the button is clicked.
4. Implementing game logic: We implement the game's logic by defining methods that get called when a button is clicked. These methods check for valid moves, change the player turn after each move, check for wins and draws, and display messages when the game is over.
5. Creating a pop-up window: When the game is over, we create a pop-up window that displays a message indicating the winner or a draw message along with a "Play Again" button.
6. Resetting the game: When the "Play Again" button is clicked, the game board is reset to its initial state, and the pop-up window is destroyed.
7. Mainloop: Finally, we call the `mainloop` method of the Tkinter window to run the game until the window is closed by the user.

By following these steps, we can create a functional Tic Tac Toe game

using Python's Tkinter library.

CODING AND TESTING

```
import tkinter as tk
class TicTacToe:
    def __init__(self, master):
        self.master = master
        self.master.title("Tic Tac
        Toe") self.turn = 'X'
        self.board = [['', '', ''], ['', '', ''], ['', '', '']]
        self.buttons = [[None, None, None], [None, None, None], [None, None, None]]
        self.create_board()

    def create_board(self):
        for row in range(3):
            for col in range(3):
                button = tk.Button(self.master, text='', font=('Arial', 60), width=4, height=2, command=lambda
row=row, col=col: self.button_click(row, col))
                button.grid(row=row,
                column=col)
                self.buttons[row][col] = button

    def button_click(self, row,
        col): if self.board[row][col]
        == '':
            self.buttons[row][col].config(text=self.turn)
            self.board[row][col] = self.turn
            if self.check_win():
                self.show_win_message()
            elif self.check_draw():
                self.show_draw_message()
            else:
                self.change_turn()

    def change_turn(self):
        if self.turn == 'X':
            self.turn = 'O'
        else:
            self.turn = 'X'

    def check_win(self):
        for i in range(3):
            if self.board[i][0] == self.board[i][1] == self.board[i][2] !=
                '': return True
            if self.board[0][i] == self.board[1][i] == self.board[2][i] !=
                '': return True
        if self.board[0][0] == self.board[1][1] == self.board[2][2] != '':
            return True
        if self.board[0][2] == self.board[1][1] == self.board[2][0] != '':
            return True
        return False

    def check_draw(self):
        for row in
        range(3):
            for col in range(3):
                if self.board[row][col] == '':
                    return False
        return True

    def show_win_message(self):
        winner = self.turn
```

```
message = f'{winner} wins!'
```

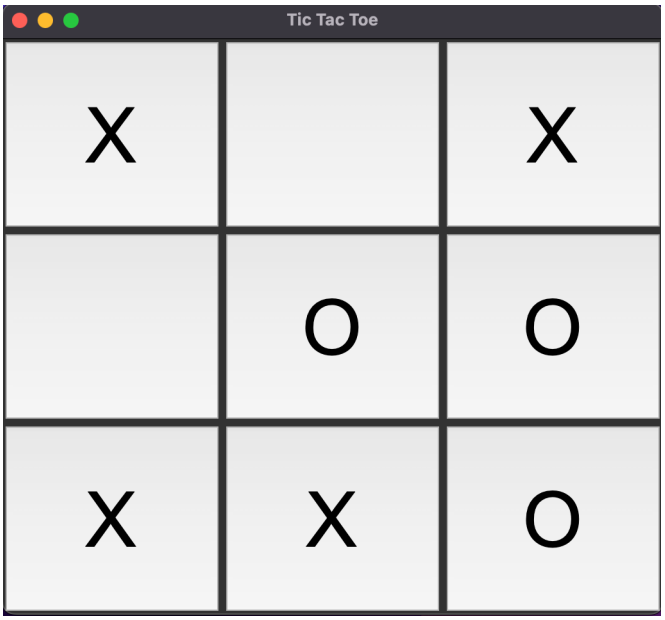
```
        self.disable_buttons()
        self.show_message(message)

    def show_draw_message(self):
        message = "It's a draw!"
        self.disable_buttons()
        self.show_message(message)

    def show_message(self, message):
        self.popup = tk.Toplevel()
        self.popup.title('Game Over')
        label = tk.Label(self.popup, text=message, font=('Arial', 24), width=10, height=2)
        label.pack(padx=10, pady=10)
        button = tk.Button(self.popup, text='Play Again', font=('Arial', 14),
        command=self.reset_board) button.pack(padx=10, pady=10)

    def reset_board(self):
        for row in
        range(3):
            for col in range(3):
                self.board[row][col] = ''
                self.buttons[row][col].config(text=''
                )
                self.buttons[row][col].config(state='normal')
        self.turn = 'X'
        self.popup.destroy()

    def disable_buttons(self):
        for row in range(3):
            for col in range(3):
                self.buttons[row][col].config(state='disabled')

root = tk.Tk()
game =
TicTacToe(root)
root.mainloop()
```

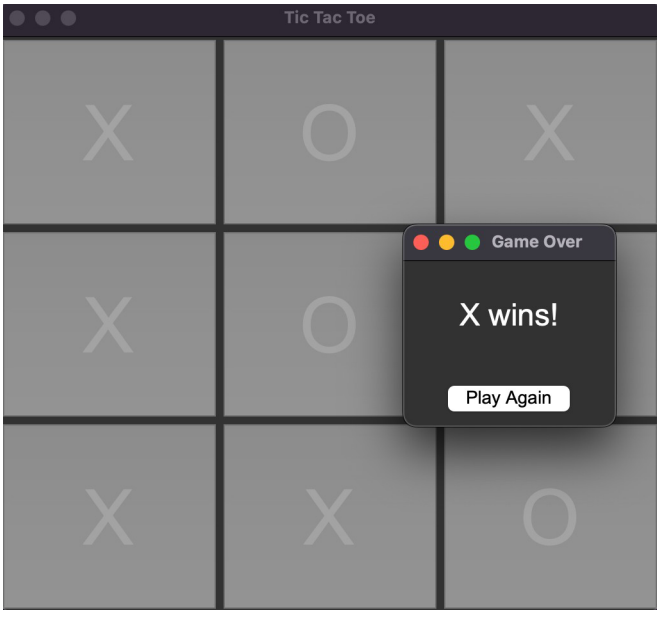
```python
import tkinter as tk

class TicTacToe:
    def __init__(self, master):
        self.master = master
        self.master.title("Tic Tac Toe")
        self.turn = 'X'
        self.board = [['', '', ''], ['', '', ''], ['', '', '']]
        self.buttons = [[None, None, None], [None, None, None], [None, None, None]]
        self.create_board()

    def create_board(self):
        for row in range(3):
            for col in range(3):
                button = tk.Button(self.master, text='', font=('Arial', 60), width=4, height=2, command=lambda row=row, col=col: self.button_click(row, col))
                button.grid(row=row, column=col)
                self.buttons[row][col] = button

    def button_click(self, row, col):
        if self.board[row][col] == '':
            self.buttons[row][col].config(text=self.turn)
            self.board[row][col] = self.turn
            if self.check_win():
                self.show_win_message()
            elif self.check_draw():
                self.show_draw_message()
            else:
                self.change_turn()

    def change_turn(self):
        if self.turn == 'X':
            self.turn = 'O'
        else:
            self.turn = 'X'

    def check_win(self):
        for i in range(3):
            if self.board[i][0] == self.board[i][1] == self.board[i][2] != '':
                return True
            if self.board[0][i] == self.board[1][i] == self.board[2][i] != '':
                return True
        if self.board[0][0] == self.board[1][1] == self.board[2][2] != '':
            return True
        if self.board[0][2] == self.board[1][1] == self.board[2][0] != '':
            return True
        return False

    def check_draw(self):
        for row in range(3):
            for col in range(3):
                if self.board[row][col] == '':
                    return False
        return True

    def show_win_message(self):
        winner = self.turn
        message = f'{winner} wins!'
        self.disable_buttons()
        self.show_message(message)

    def show_draw_message(self):
        message = "It's a draw!"
        self.disable_buttons()
        self.show_message(message)

    def show_message(self, message):
        self.popup = tk.Toplevel()
        self.popup.title('Game Over')
        label = tk.Label(self.popup, text=message, font=('Arial', 24), width=10, height=2)
        label.pack(padx=10, pady=10)
        button = tk.Button(self.popup, text='Play Again', font=('Arial', 14), command=self.reset_board)
        button.pack(padx=10, pady=10)

    def reset_board(self):
        for row in range(3):
            for col in range(3):
                self.board[row][col] = ''
                self.buttons[row][col].config(text='')
                self.buttons[row][col].config(state='normal')
        self.turn = 'X'
        self.popup.destroy()

    def disable_buttons(self):
        for row in range(3):
            for col in range(3):
                self.buttons[row][col].config(state='disabled')

root = tk.Tk()
game = TicTacToe(root)
root.mainloop()
```

OUTPUT AND RESULTS

Results:- Tic Tac Toe using Python Tkinter was successfully implemented and executed. Screenshots of the output has been attached below.

OUTPUT:





CONCLUSION AND FUTURE ENHANCEMENT

In conclusion, we have successfully implemented the classic Tic Tac Toe game using Python's Tkinter library, providing users with an interactive and engaging way to play the game on their computers. Through the `TicTacToe` class and associated methods, we have managed the game's state and logic, allowing for a smooth and enjoyable gaming experience.

As for future enhancements, we can consider adding more features to the game, such as:

1. Difficulty levels: We can add difficulty levels to the game to challenge players with different levels of experience. For instance, we can create an AI-based player that can be programmed to play at varying levels of difficulty.
2. Multiplayer mode: We can add a multiplayer mode that allows players to compete with each other locally or over the internet.
3. Visual enhancements: We can enhance the game's graphics and add more animations and sound effects to make the game more appealing and immersive.
4. Leaderboards: We can add a leaderboard feature that tracks the top scores of players, creating a competitive element to the game.

Overall, with the addition of these features, we can make the game more engaging and competitive, providing users with a more enjoyable gaming experience.

REFERENCES

1. Pythons official documentation:
https://docs.python.org/3/tutorial/index.htm
l
2. TkDocs: http://www.tkdocs.com/tutorial/index.html
3. Python for Beginners: https://www.pythonforbeginners.com/
4. Programiz: https://www.programiz.com/python-programming
5. tkinter 8.5 reference: https://tkdocs.com/tutorial/index.html