Data Structures & Algorithm- 3CS507C24

Special assignment report on

Tic Tac Toe Implementation Using Queue

Submitted to :- Prof. Ashwin Verma

Submitted by :-

22BEI039 - Khushal Modha

Introduction :-

This project report presents the development of a Tic Tac Toe game using Python's Tkinter library for the graphical user interface and the Queue module for managing game states and player turns. The objective was to create a simple yet interactive two-player game that captures the essence of the classic Tic Tac Toe experience. The implementation involves setting up a 3x3 grid, handling player moves, checking for win conditions, and displaying the game outcome. The application was tested for functionality, confirming accurate move tracking and win detection. Future enhancements include adding an AI opponent, a reset feature, and improving the user interface. This project serves as an educational exploration of GUI programming and game development in Python.

Purpose of the Project :-

The queue data structure was selected for this Tic-Tac-Toe implementation due to its ease of use and effectiveness, particularly in handling player turns and move validation. While other data structures, such as stacks, might also be used, doing so would complicate implementation since stacks would need to be managed multiple times, one for each game situation (rows, columns, and diagonals). On the other hand, queues expedite the procedure by making the diagonal validation—which is crucial in Tic-Tac-Toe—less complicated. Because of the First-In-First-Out (FIFO) structure of the queue, player turns can alternate in a predictable way. Furthermore, this decision is in line with upcoming implementations, especially when expanding the game's dimensions beyond the conventional 3x3 grid. A queue is the best option for current and future project extensions because it will continue to be easier to manage larger grids than a stack-based method.

Problem Statement :-

Tic-Tac-Toe is a simple two-player game played on a 3x3 grid.Each player uses the sign 'X' or 'O' to mark one of the empty spots on the grid, then the other player does the same. Being the first player to line up three of their symbols in a row—horizontally, vertically, or diagonally—is the aim of the game. In the event that neither player fills every spot on the grid, the game is a draw. The game is designed to be quick and strategic, with every action having the ability to affect the options available to the opposition.

Challenges :-

There are numerous difficulties when implementing Tic-Tac-Toe with a queue because of the structure of the queue data structure. According to the First-In-First-Out (FIFO) rule, the person who joins the queue first will always go first and is positioned at the rear of the queue following their move. While this approach is effective at handling player turns that alternate, it can cause issues with the general logic of the game, including:

Managing Player Turns:

The main difficulty is finding a fair approach to swap turns utilizing a queue. The game must effectively dequeue a player after their move and re-queue them at the rear for the following round in order to track and manage each player's turn within the queue.

Validating Winning Conditions:

Keeping track of players' progress through the

queue while keeping an eye out for winning criteria (three symbols in a row) is another difficulty. The positions of the 'X's and 'O's on the grid are not automatically tracked by the queue, therefore extra logic is required to verify the rows, columns, and diagonals following each move. It's crucial to make sure that this logic works well with queue management in order to keep the game flow simple.

Diagonal Validation:

Because the positions are not continuous in the grid, checking diagonals is especially difficult. To make matters more complicated, keeping track of the positions requires utilizing a queue. In contrast to other structures like stacks, queues make this procedure simpler by making it easy to handle sequences for validation.

Overview of Queue :-

First-In-First-Out (FIFO) dictates that the first element added to a queue is the first one withdrawn, making a queue a linear data structure. This kind of behavior is comparable to standing in line for services in the real world. When elements are

arranged in a queue, new ones are added at the back and old ones are taken out front. In scenarios like scheduling or task management, when an activity or task must be completed in a particular order, queues are often utilized.

Justification of Using Queue :-

The queue data structure was selected for this Tic-Tac-Toe implementation because of its FIFO trait, which makes alternating turns between players easier. Player movement results in their placement at the rear of the queue, which eliminates the need for manual tracking and guarantees smooth turn transitions.

Additionally, queues make game concepts like diagonal validation simpler. This is because stacks make it more difficult to manage various stacks for rows, columns, and diagonals. Additionally, because queues scale readily without requiring significant changes to the basic logic, they are future-proof for larger grids (e.g., 4x4 or 5x5). In general, player management and game validation are more scalable and efficient when a queue is used.

Pseudocode :-

1. Initialize Game
    - Set player "X" as the starting player.

- Create a 3x3 grid and queues for rows, columns, and diagonals.
- Display the home page.

2. Home Page
   - Clear the current frame.
   - Display title and game options.
   - Show the winner or draw message if applicable.
   - Add a "Play Game" button to start the game.
3. Start Game
   - Clear frame, reset game state (board, queues, turn, moves), and create the 3x3 game board.
4. On Button Click
   - Place player's symbol if the cell is empty.
   - Update the board and queues for the move.
   - Check for a winner or draw. If found, declare it.
   - Switch turns between "X" and "O".
5. Update Queues
   - Add the player’s symbol to the relevant row, column, and diagonal queues.
6. Check Winner
   - Check if any row, column, or diagonal queue has three matching symbols.
7. Declare Winner/Draw
   - Display the winner or draw message and return to the home page.
8. Exit Fullscreen
   - Exit fullscreen when 'Esc' is pressed.
9. Main Program
   - Initialize the Tic-Tac-Toe game and start the main loop.

Prerequisites for Running the Tic-Tac-Toe Game :-

1. Python Installation:

- You must have Python installed on your system. Download and install Python from the official website: https://www.python.org/downloads/. **2. Tkinter (Python's Built-in GUI Library):**
  - Tkinter is included with Python by default. However, if it's not installed or enabled on your system, you can install it using the following command:

```
apt-get install python3-tk
```

3. Queue Module:

- The queue module is part of Python's standard library, so no additional installation is required.

4. Running the Code:

- Save the provided code as a Python (.py) file.
- Run the file using Python from your terminal or command prompt: Copy code

```
python tic_tac_toe.py
```

Conclusion :-

In conclusion this version of Tic Tac Toe shows how to manage player turns and validate game states by using the queue data structure effectively. The game makes use of queues' FIFO nature to provide seamless player rotation without the need for intricate tracking systems. Furthermore, queues streamline the administration of row, column, and diagonal validations, resulting in more scalable and efficient game logic—particularly for potential future expansions to larger grids. This project demonstrates how data structures can be used practically to address typical game development problems while still meeting the assignment's requirements.