Final Project Report

Dr. Strange's Tic Tac Toe

Submitted by:

Pedro Cabrera

Course Professor:

Prof. J. Medina

COP – 3275C

C/C++ Programming for Engineers

Date Report Submitted:

04/23/2024

Engineering, Computer Programming, And Technology

Department of Electrical and Computer Engineering Technology – ECET

Table of Contents

Purpose

The objective of this project is to showcase the practical application of C++ programming skills acquired during the course. By integrating concepts from each chapter, this program synthesizes a unique and interactive Tic Tac Toe game, "Dr. Strange's Tic Tac Toe." It features an intelligent AI opponent nicknamed "Dr. Strange," capable of predictive and strategic play, offering users both a single-player and a competitive two-player experience. The program further stands out by incorporating a ScoreBoard class that tracks and saves game results, enabling players to review their performance history and observe their improvement over time.

Description

"Dr. Strange's Tic Tac Toe" represents a sophisticated version of the traditional game by integrating an intelligent AI opponent. Named after the formidable Marvel character Dr. Strange, the AI showcases its powers through a strategic analysis reminiscent of the character's ability to foresee multiple futures. This AI employs a streamlined version of the minimax algorithm, which allows it to evaluate the board's state and decide on the most advantageous moves to either secure a win or block the player's progress.

The program initiates by offering players the choice to load previous game scores from a file or begin a new game, enhancing user engagement through continuity and score tracking. This feature utilizes file input/output operations extensively to manage game data, allowing for ongoing tracking of player achievements across multiple sessions.

Once the game starts, players can choose between competing against the AI in a single-player mode or engaging in a two-player mode where two players can challenge each other. In single-player mode, the AI, Dr. Strange, calculates its moves based on the current board layout, prioritizing offensive strategies to clinch a victory or defensive tactics to thwart the player's chances. In contrast, the two-player mode operates on a turn-by-turn basis without AI involvement, focusing instead on player interaction and decision-making.

The game dynamically displays the board in the console, updating it after each move to ensure clarity and ease of use. The board itself is managed through a vector of vectors, which records the positions of 'X' and 'O' markers. This setup not only facilitates an intuitive gameplay experience but also allows for easy modifications and updates to the game logic.

Core functionalities of the program include move validation to ensure that all moves are legal, a comprehensive check for win conditions after each move, and a robust score tracking system that records wins, losses, and ties. Each game outcome is meticulously logged by the ScoreBoard class, which updates the cumulative results continuously and saves them to a designated file.

The user interface loops back to the main menu after each game, providing options to start a new game, view the scoreboard, or exit. This loop ensures that players can engage with the game repeatedly, fostering an environment of continuous interaction and strategic challenge.

List of Functions Used:

Chapter 2 - Problem Solving Using C++

- **Variable declaration statements**: Variables are declared across various classes.
- **cout statements**: Used extensively for outputting game status and instructions.
- **Arithmetic operations**: Employed in score calculations and game logic.

Chapter 3 - Assignment, Formatting, and Interactive Input

- **Assignment operations**: Assignments to variables and class members throughout the code.
- **Accumulating operations**: Incrementing scores in the **recordWin** and **recordTie** methods.
- **Program input using cin**: Collects user inputs to determine game moves and settings.

Chapter 5 - Repetition Statements

- **While loops**: Managing game sessions and checking game conditions.
- **For loops**: To run through all the values of 2D Array/Vector.

Chapter 6 - Modularity Using Functions

- **Function Prototypes**: All class methods are prototyped and defined.
- **Calling a function**: Functions are called to operate the game logic and UI interactions.

Chapter 7 - Arrays

- **Declaring and Processing Two-Dimensional Arrays:** Though my code uses std::vector for a dynamic array, it functions similarly to a two-dimensional array in handling the game board.
- **The Standard Template Library (STL):** I've employed the vector class from the STL to create a flexible and dynamic game board. **Table (7.2)**

- **Array Initialization:** The game board vector is initialized with spaces representing an empty board.

Chapter 8 - I/O Streams and Data Files

- **File stream objects**: Utilized to handle reading from and writing to files.
- **Opening output files for writing**: Executed in the **saveScores** method.
- **Opening input files for reading**: Implemented in the **loadScores** method.

Chapter 9 - String class

- Uses the **string** class to manage text data like file names.

Chapter 10 - Pointers and Dynamic Data Structures

- **Pointers**: Employed in the **recordScore** method where pointers to integer member variables of the **ScoreBoard** class are used to manipulate game scores.

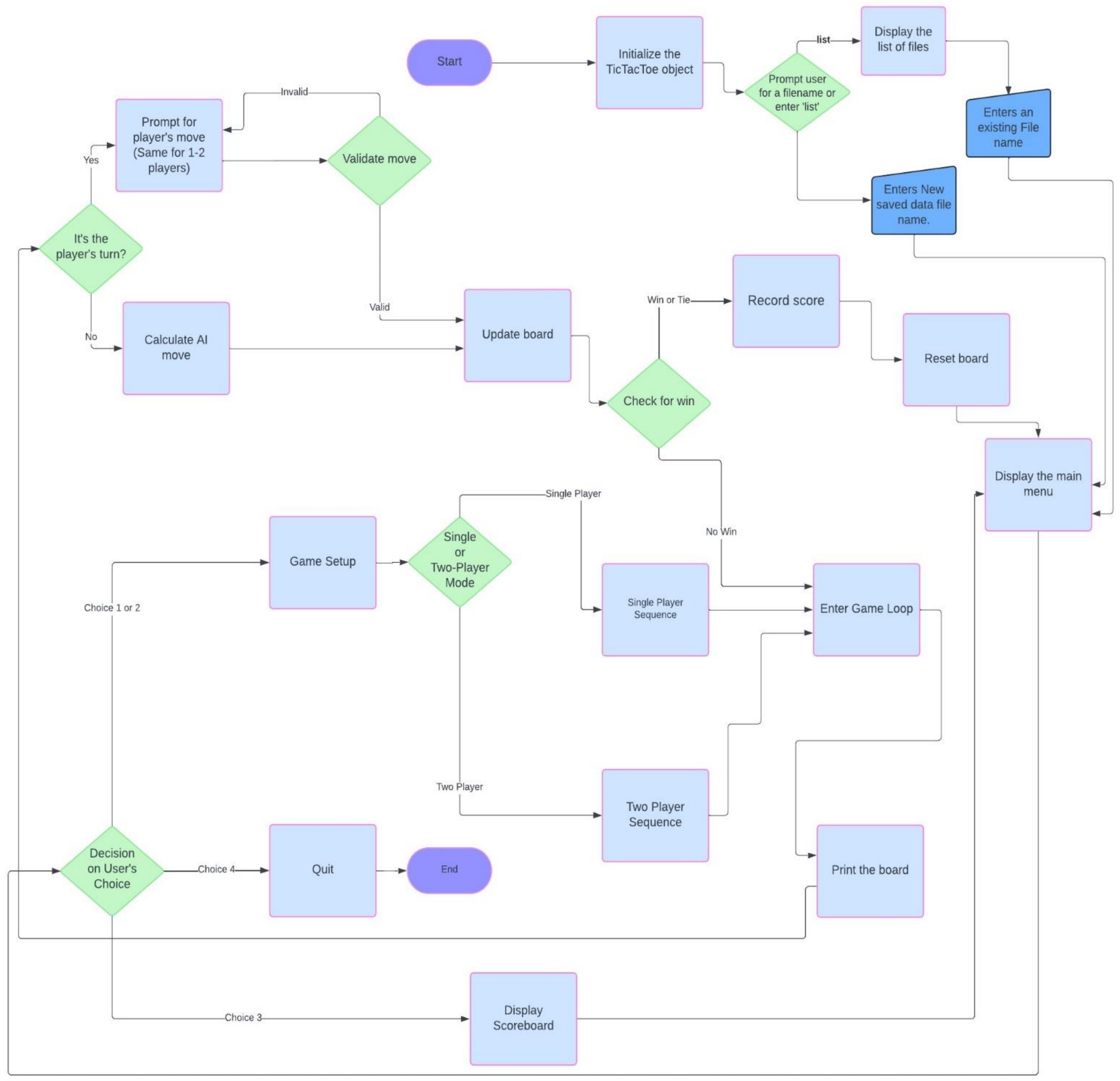
Chapter 11 - Classes

- **Class declaration and implementation**: Organizing game data and functions into logical units (**ScoreBoard** and **TicTacToe**).

Chapter 12 - Adding Functionality to Your Classes

- **Class I/O Capabilities:** My ScoreBoard class incorporates I/O functionalities as it reads and writes scores to files.

Flowchart:



**The Flowchart was made in a snail like pattern for compactness and to better describe the functionality of the code. **