**NAME:**

MARC JYSK E. HERATO / JASMINE ALEXA C. TORENTE / ERIKA ALEXI M. MAGSINO

**COURSE/YEAR/BLOCK:** BSCS-2B

**COURSE SUBJECT:** DATA STRUCTURES AND ALGORITHMS

**INSTRUCTOR:** KHRISTINE BOTIN

**FINAL PROJECT / EXAMINATION**

**Project Overview**

This project is a game of **Tic Tac Toe** in C++ language with enhanced functionalities. The game uses data structures such as **Binary Trees**, **Binary Search Trees (BST)**, and **Heaps** to model and manage different aspects of the game. It allows two players to alternate turns placing their marks ('X' or 'O') on a 3x3 grid, with the goal of achieving three consecutive marks in a row, column, or diagonal.

Key features include:

* A **binary tree** representing the game board.
* A **binary search tree** validating valid player moves.
* A **max-heap** to manage and display player scores efficiently.
* Player turn alternation, win condition checking, tie tracking, and score management.

**How to Run the Code**

**Prerequisites**

* A C++ compiler supporting C++11 or later (e.g., GCC, Clang, or MSVC).
* An IDE or terminal for compiling and running the program.

**Steps to Compile and Execute**

1. Copy the source code into a file, e.g., TicTacToe.cpp.
2. Open a terminal or IDE.
3. Compile the code:
4. Run the program:
5. Follow the on-screen instructions to play the game.

**Description of Each Functionality**

**Binary Tree: Game Board Representation**

* The **TicTacToeTree** class represents the **3x3 board** using a binary tree.
  + Each node corresponds to a cell (positions 1–9).
  + Example tree structure:

**5**

**/ \**

**3 7**

**/ \ / \**

**1 4 6 9**

**\ /**

**2 8**

* **Key Methods**:
  + buildTree(): Constructs the tree by linking nodes to represent positions.
  + displayBoard(): Performs a **level-order traversal** to populate a 3x3 grid and display it in a user-friendly format.
  + makeMove(): Recursively searches the tree to update the symbol ('X' or 'O') for a given position, ensuring no overwriting of existing moves.
  + resetBoard(): Resets all nodes’ symbols to ' ' for a new game.

**Binary Search Tree: Move Validation**

* The **MoveValidator** class uses a BST to validate moves.
  + **Purpose**: Ensure moves are within bounds (1–9) and not repeated.
  + Positions 1–9 are preloaded into the BST during initialization.
  + **Key Methods**:
    - insert(int pos): Adds a position to the BST.
    - search(int pos): Verifies if the position exists in the BST and is valid.

**Max-Heap: Score Management**

* The **MaxHeap** class manages player scores using a **priority queue** (a max-heap).
  + Tracks scores for Player X, Player O, and the count of ties.
  + Swap the players position depending on the highest score.
  + **Key Methods**:
    - updateScore(char player): Updates the score for the winner of a round.
    - updateTie(): Increments the tie count.
    - displayScores(): Prints current scores and the highest score.
    - heapifyUp(): Swap the players position upward.
    - heapifyDown(): Swap the players position downward.

**Game Logic: Main Gameplay**

* The **TicTacToeGame** class contains the main game logic:
  + Initializes the board, move validator, and score heap.
  + Implements the **game loop**, allowing players to alternate turns, input moves, and check for wins or ties.
  + **Key Methods**:
    - playGame(): Runs the main game loop.
    - checkWin(char player): Evaluates all possible winning combinations using predefined position arrays.
    - resetGame(): Clears the board and resets the move count for a new round.
    - checkPosition(): Recursively checks if a specific position matches the player’s symbol.

**Advanced Functionalities and Features**

**Win/Tie Detection**

* Winning combinations (rows, columns, diagonals) are stored in a 2D array:

int winCombos[8][3] = {

{1, 2, 3}, {4, 5, 6}, {7, 8, 9}, // Rows

{1, 4, 7}, {2, 5, 8}, {3, 6, 9}, // Columns

{1, 5, 9}, {3, 5, 7} // Diagonals

};

* checkWin() evaluates each combination to determine if the player has won.

**Efficient Score Retrieval**

* The use of a **max-heap** ensures efficient tracking of the highest scores:
  + Adding a new score
  + Retrieving the top score

**Replay Capability**

* Players can replay rounds, with scores persisting across games, ensuring a seamless user experience.

**Example Gameplay**

**Game Start**

Welcome to Tic Tac Toe!

1 | 2 | 3

---+---+---

4 | 5 | 6

---+---+---

7 | 8 | 9

**Player Moves**

* **Player X enters 5**:

1 | 2 | 3

---+---+---

4 | X | 6

---+---+---

7 | 8 | 9

* **Player O enters 1**:

O | 2 | 3

---+---+---

4 | X | 6

---+---+---

7 | 8 | 9

**End of Game**

* If Player X wins:

Player X wins this round!

Scores:

Player X: 1

Player O: 0

Tie: 0

**Replay or Exit**

Do you want to play another round? (y/n):

**Github link**

https://github.com/Maruku-lazy/Final-Project-Examination-DSA