Tic Tac Toe Game Report

Introduction:

The provided code implements a Tic Tac Toe game in C++. It allows both single-player and double-player modes, where players can compete against each other or play against an AI computer opponent. The game utilizes the minimax algorithm to determine the best moves for the computer player.

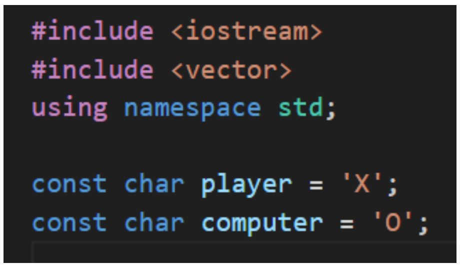
Code Structure:

1. Libraries and Namespace:

-Iostream and Vector Libraries are used for input/output operations and using vectors, respectively.

- The "std" namespace is used to simplify the code by allowing direct access to standard library functions and objects without prefixing them with "std::".

- Two constant characters, 'X' and 'O', are defined to represent the player and computer markers, respectively.

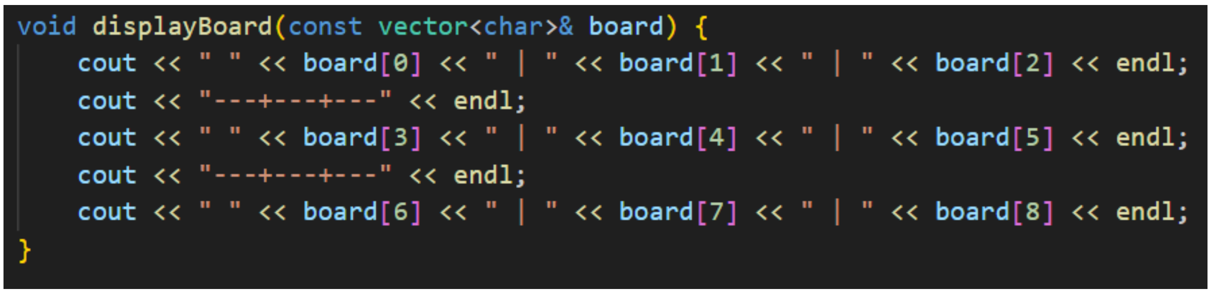


```cpp
#include <iostream>
#include <vector>
using namespace std;

const char player = 'X';
const char computer = 'O';
```

2. Display Board Function:

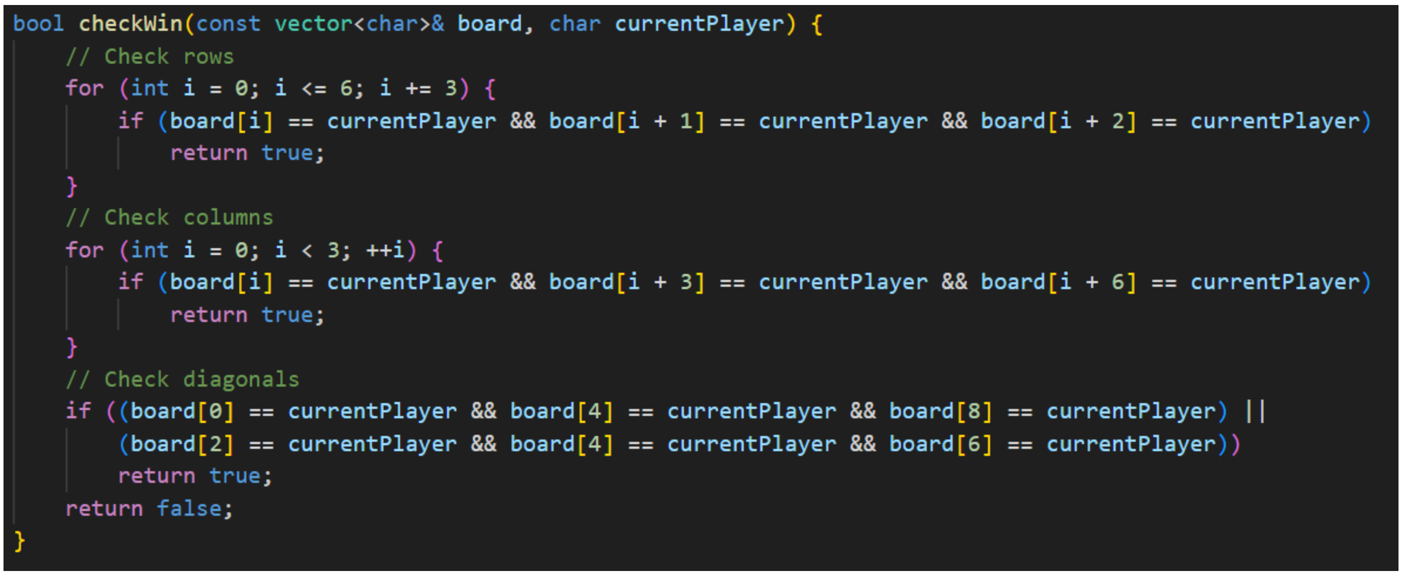
- The "displayBoard" function takes a vector representing the game board as a parameter and prints it in the console with a visual representation of the game grid.



```cpp
void displayBoard(const vector<char>& board) {
    cout << " " << board[0] << " | " << board[1] << " | " << board[2] << endl;
    cout << "---+---+---" << endl;
    cout << " " << board[3] << " | " << board[4] << " | " << board[5] << endl;
    cout << "---+---+---" << endl;
    cout << " " << board[6] << " | " << board[7] << " | " << board[8] << endl;
}
```

3. Check Win Function:

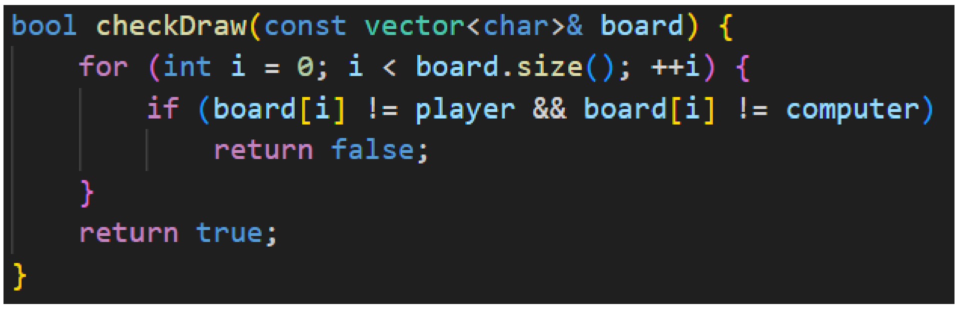
- The "checkWin" function checks if a player has won the game by examining the rows, columns, and diagonals of the game board. If all the cells of a particular row, column or diagonal are filled by the current player’s symbol, then the current Player is declared as winner. This function is called after each Player’s turn.



```cpp
bool checkWin(const vector<char>& board, char currentPlayer) {
    // Check rows
    for (int i = 0; i <= 6; i += 3) {
        if (board[i] == currentPlayer && board[i + 1] == currentPlayer && board[i + 2] == currentPlayer)
            return true;
    }
    // Check columns
    for (int i = 0; i < 3; ++i) {
        if (board[i] == currentPlayer && board[i + 3] == currentPlayer && board[i + 6] == currentPlayer)
            return true;
    }
    // Check diagonals
    if ((board[0] == currentPlayer && board[4] == currentPlayer && board[8] == currentPlayer) ||
        (board[2] == currentPlayer && board[4] == currentPlayer && board[6] == currentPlayer))
        return true;
    return false;
}
```

4. Check Draw Function:

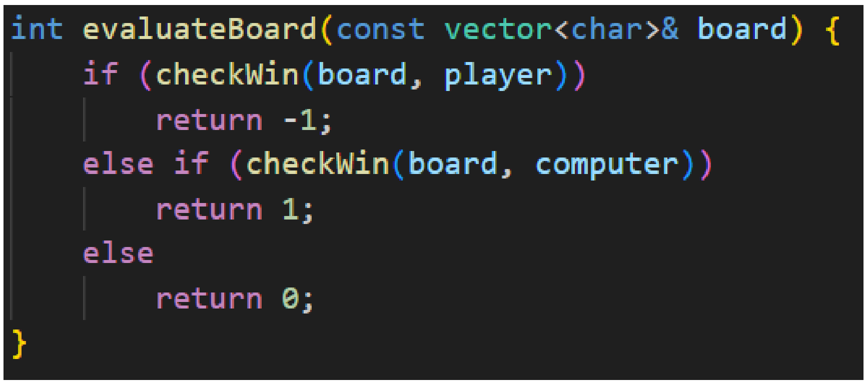
- The "checkDraw" function checks if the game ends in a draw by verifying if all the cells on the board are filled.



```cpp
bool checkDraw(const vector<char>& board) {
    for (int i = 0; i < board.size(); ++i) {
        if (board[i] != player && board[i] != computer)
            return false;
    }
    return true;
}
```

5. Evaluate Board Function:

- The "evaluateBoard" function evaluates the current state of the game board and returns a score (-1, 0, 1) indicating whether the player has won, lost, or the game is still in progress. Depending on the current state, this function decides whether the functions for taking input have to be initiated again or the game has ended.



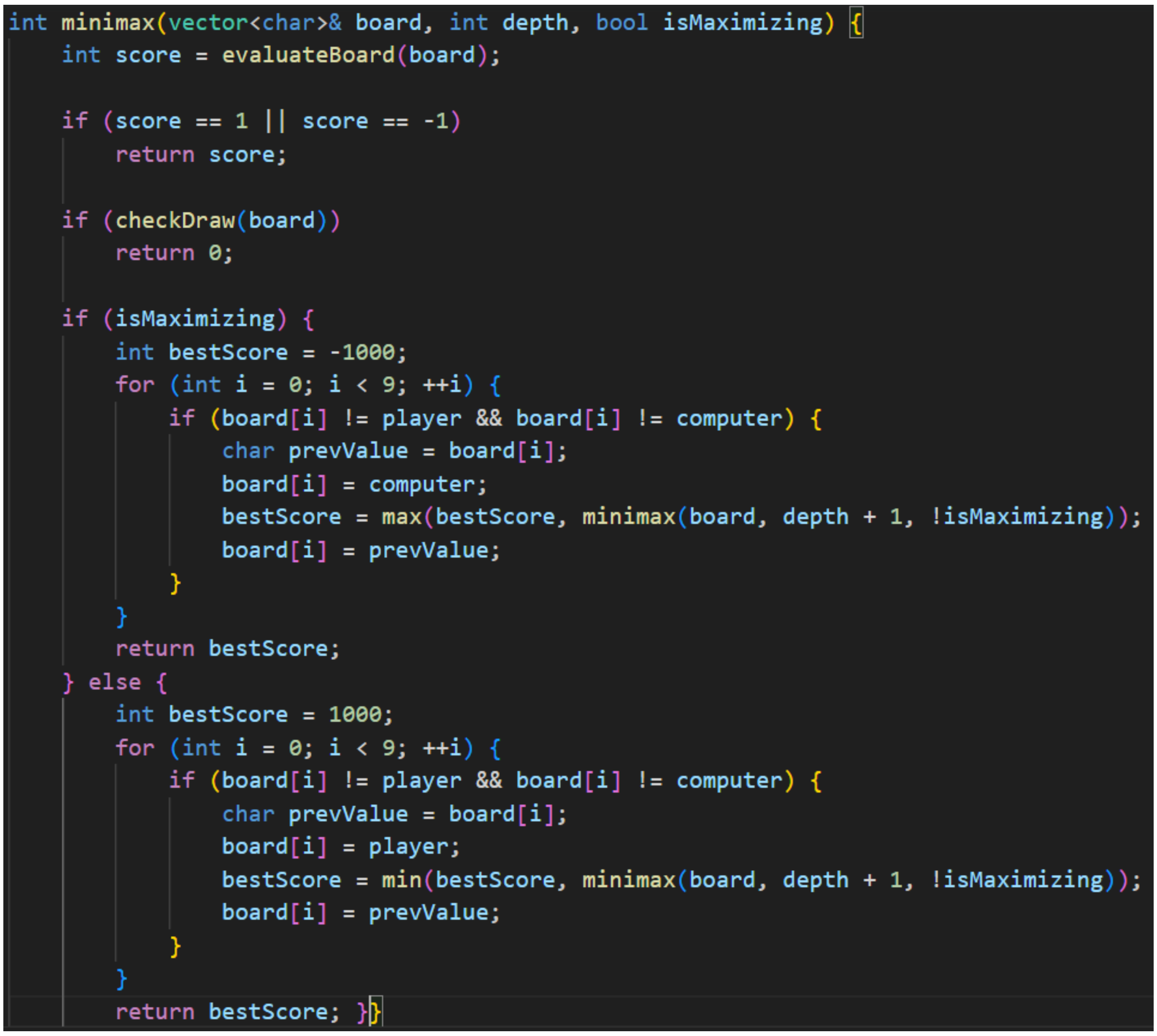
```cpp
int evaluateBoard(const vector<char>& board) {
    if (checkWin(board, player))
        return -1;
    else if (checkWin(board, computer))
        return 1;
    else
        return 0;
}
```

6. Minimax Algorithm and Best Move Function:

- The "minimax" function implements the minimax algorithm, which recursively determines the best score for each possible move, considering the current player and opponent. If it's the maximizing player's turn, the function initializes bestScore to a very low value (-1000). It iterates over all available moves on the board and simulates each move by placing the computer player's marker in an empty cell. It then recursively calls the minimax function with the updated board, increased depth, and switched turn. The best score is updated with the maximum value between the current bestScore and the score obtained from the recursive call. The function reverts the move by restoring the original value of the cell on the board. Finally, it returns the bestScore.

If it's the minimizing player's turn, the function follows a similar process as the maximizing case, but initializes bestScore to a high value (1000) instead. The goal is to minimize the score in this case.

- It uses the "evaluateBoard" function to assign scores to terminal states and backtracks to find the optimal move for the computer player.



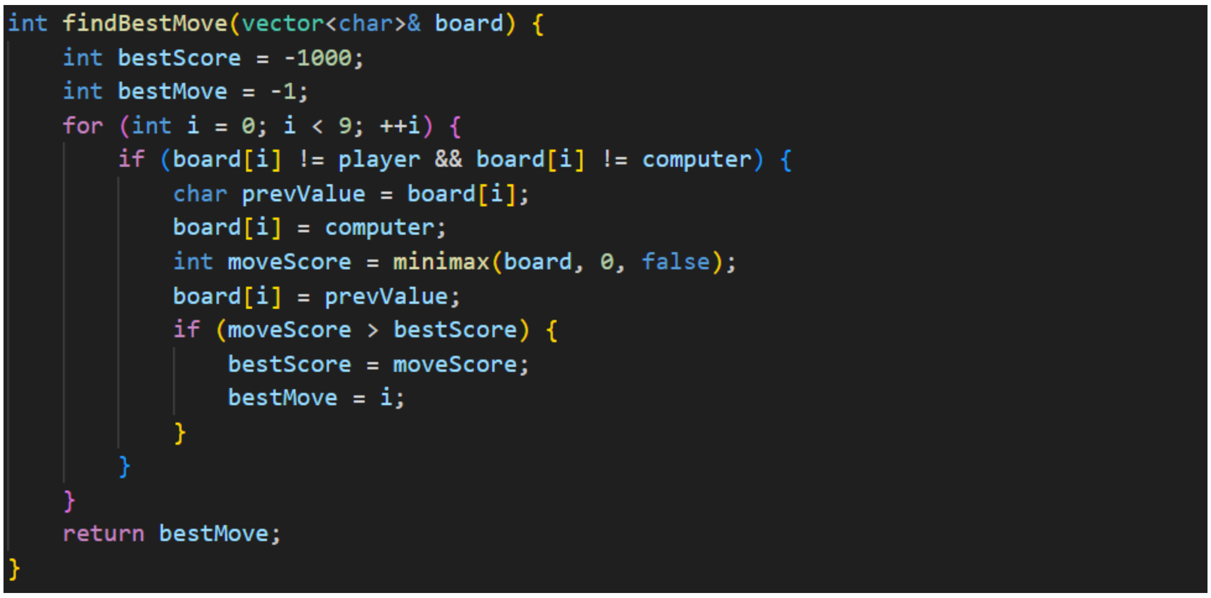
```
int minimax(vector<char>& board, int depth, bool isMaximizing) {
    int score = evaluateBoard(board);

    if (score == 1 || score == -1)
        return score;

    if (checkDraw(board))
        return 0;

    if (isMaximizing) {
        int bestScore = -1000;
        for (int i = 0; i < 9; ++i) {
            if (board[i] != player && board[i] != computer) {
                char prevValue = board[i];
                board[i] = computer;
                bestScore = max(bestScore, minimax(board, depth + 1, !isMaximizing));
                board[i] = prevValue;
            }
        }
        return bestScore;
    } else {
        int bestScore = 1000;
        for (int i = 0; i < 9; ++i) {
            if (board[i] != player && board[i] != computer) {
                char prevValue = board[i];
                board[i] = player;
                bestScore = min(bestScore, minimax(board, depth + 1, !isMaximizing));
                board[i] = prevValue;
            }
        }
        return bestScore; }}
```

The findBestMove function utilizes the minimax function to find the best move for the computer player. It iterates over all available moves, simulates each move, and calls the minimax function to obtain the score for that move. After evaluating all possible moves, it returns the index of the move with the highest score (bestMove)



```cpp
int findBestMove(vector<char>& board) {
    int bestScore = -1000;
    int bestMove = -1;
    for (int i = 0; i < 9; ++i) {
        if (board[i] != player && board[i] != computer) {
            char prevValue = board[i];
            board[i] = computer;
            int moveScore = minimax(board, 0, false);
            board[i] = prevValue;
            if (moveScore > bestScore) {
                bestScore = moveScore;
                bestMove = i;
            }
        }
    }
    return bestMove;
}
```

7. Player and Computer Turns:

- The "playerTurn" function allows the human player to make a move by selecting an empty cell on the board and the "computerTurn" function determines the best move for the computer player using the "findBestMove" function.

- The "playGame_singleplayer" function represents the single-player game mode.

- The "playGame_doubleplayer" function represents the double-player game mode.

8. Main Function:

- The "main" function serves as the entry point of the program.

- It prompts the user to choose between single-player and double-player modes and calls the respective function based on the input.

- After each game, it asks the user if they want to play again and continues accordingly.

Report Conclusion:

The provided code implements a functional Tic Tac Toe game with options for both single-player and double-player modes. It utilizes the minimax algorithm to provide a challenging AI opponent in the single-player mode. The code is well-structured, modular, and follows standard C++ conventions. Players can enjoy the game by running the program, selecting the desired mode, and playing against either the computer or another player.