Name: Shabbar Adamjee  
Roll No.: PB57  
PRN: 1032221508

*AIES ASSIGNMENT 2*

*MINIMAX ALGORITHM – TICTACTOE*

*Code*

#include <array>

#include <iostream>

#include <limits>

typedef std::array<std::array<char, 3>, 3> matrix;

void miniMax(matrix &gameBoard, char currentPlayer);

int maxValue(matrix &gameBoard);

int minValue(matrix &gameBoard);

bool checkFilledBoard(const matrix &gameBoard);

std::pair<char, bool> checkWinCondition(const matrix &gameBoard);

void miniMax(matrix &gameBoard, char currentPlayer) {

  int bestValue = (currentPlayer == 'X')

                      ? std::numeric\_limits<int>::min()  // -2147483648

                      : std::numeric\_limits<int>::max(); // 2147483647

  int bestMoveRow = -1, bestMoveCol = -1;

  // Traverse all cells to find the best move

  for (int i = 0; i < 3; i++) {

    for (int j = 0; j < 3; j++) {

      if (gameBoard[i][j] == '\0') {

        gameBoard[i][j] = currentPlayer; // Apply move X or O

        int moveValue =

            (currentPlayer == 'X') ? minValue(gameBoard) : maxValue(gameBoard);

        gameBoard[i][j] = '\0'; // Reset

        // Choose the best move for X or O

        if ((currentPlayer == 'X' && moveValue > bestValue) ||

            (currentPlayer == 'O' && moveValue < bestValue)) {

          bestMoveRow = i;

          bestMoveCol = j;

          bestValue = moveValue;

        }

      }

    }

  }

  // Apply the best move

  if (bestMoveRow != -1 && bestMoveCol != -1) {

    gameBoard[bestMoveRow][bestMoveCol] = currentPlayer;

    std::cout << "Best move for " << currentPlayer << ": (" << bestMoveRow

              << ", " << bestMoveCol << ")\n";

  }

}

// Maximizer

int maxValue(matrix &gameBoard) {

  auto [winner, isWin] = checkWinCondition(gameBoard);

  if (isWin) {

    if (winner == 'X')

      return 10;

    if (winner == 'O')

      return -10;

  }

  if (checkFilledBoard(gameBoard))

    return 0;

  int bestValue = std::numeric\_limits<int>::min();

  for (int i = 0; i < 3; i++) {

    for (int j = 0; j < 3; j++) {

      if (gameBoard[i][j] == '\0') {

        gameBoard[i][j] = 'X'; // Maximizer's move

        bestValue = std::max(bestValue, minValue(gameBoard));

        gameBoard[i][j] = '\0'; // Undo the move

      }

    }

  }

  return bestValue;

}

// Minimizer

int minValue(matrix &gameBoard) {

  // returns a pair <char, bool>

  auto [winner, isWin] = checkWinCondition(gameBoard);

  if (isWin) {

    if (winner == 'X')

      return 10;

    if (winner == 'O')

      return -10;

  }

  if (checkFilledBoard(gameBoard))

    return 0;

  int bestValue = std::numeric\_limits<int>::max();

  for (int i = 0; i < 3; i++) {

    for (int j = 0; j < 3; j++) {

      if (gameBoard[i][j] == '\0') {

        gameBoard[i][j] = 'O'; // Minimizer's move

        bestValue = std::min(bestValue, maxValue(gameBoard));

        gameBoard[i][j] = '\0'; // Undo the move

      }

    }

  }

  return bestValue;

}

// Function to check if the board is filled

bool checkFilledBoard(const matrix &gameBoard) {

  for (int i = 0; i < 3; i++) {

    for (int j = 0; j < 3; j++) {

      if (gameBoard[i][j] == '\0')

        return false;

    }

  }

  return true;

}

// Helper function to check if a line is complete

bool checkLine(const matrix &gameBoard, int x1, int y1, int x2, int y2, int x3,

               int y3) {

  return gameBoard[x1][y1] == gameBoard[x2][y2] &&

         gameBoard[x1][y1] == gameBoard[x3][y3] && gameBoard[x1][y1] != '\0';

}

// Function to check if someone has won

std::pair<char, bool> checkWinCondition(const matrix &gameBoard) {

  // Check rows, columns, and diagonals

  if (checkLine(gameBoard, 0, 0, 0, 1, 0, 2))

    return {gameBoard[0][0], true};

  if (checkLine(gameBoard, 1, 0, 1, 1, 1, 2))

    return {gameBoard[1][0], true};

  if (checkLine(gameBoard, 2, 0, 2, 1, 2, 2))

    return {gameBoard[2][0], true};

  if (checkLine(gameBoard, 0, 0, 1, 0, 2, 0))

    return {gameBoard[0][0], true};

  if (checkLine(gameBoard, 0, 1, 1, 1, 2, 1))

    return {gameBoard[0][1], true};

  if (checkLine(gameBoard, 0, 2, 1, 2, 2, 2))

    return {gameBoard[0][2], true};

  if (checkLine(gameBoard, 0, 0, 1, 1, 2, 2))

    return {gameBoard[0][0], true};

  if (checkLine(gameBoard, 0, 2, 1, 1, 2, 0))

    return {gameBoard[0][2], true};

  return {'\0', false};

}

int main() {

  matrix gameBoard{};

  bool gameEnded = false;

  char currentPlayer = 'X'; // First player X

  while (!gameEnded) {

    // Display the board

    std::cout << "Current board state:\n";

    for (int i = 0; i < 3; i++) {

      for (int j = 0; j < 3; j++) {

        std::cout << (gameBoard[i][j] == '\0' ? '-' : gameBoard[i][j]) << " ";

      }

      std::cout << std::endl;

    }

    miniMax(gameBoard, currentPlayer);

    // Check for a win condition

    auto [winner, isWin] = checkWinCondition(gameBoard);

    if (isWin) {

      std::cout << "Player " << winner << " wins!\n";

      gameEnded = true;

      break;

    }

    // Check for a draw

    if (checkFilledBoard(gameBoard)) {

      std::cout << "It's a draw!\n";

      gameEnded = true;

      break;

    }

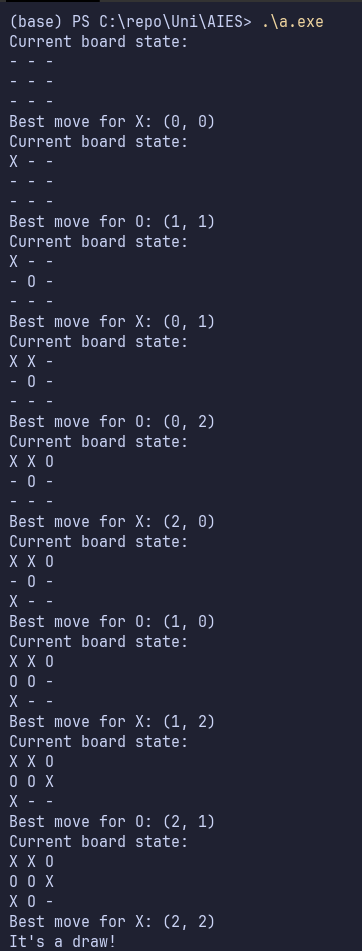
    // Switch players: 'X' -> 'O', 'O' -> 'X'

    currentPlayer = (currentPlayer == 'X') ? 'O' : 'X';

  }

  return 0;

}

*Output*