**AI LAB ASSIGNMENT 1**

**Name: Atharva Salitri PRN: 12310120**

**Div : CSAI B Batch: 2 Roll No: 37**

**1) Implement Tic tac toe without using AI (with Draw guessing logic)**

**Code:**

#include <stdio.h>

#include <stdlib.h>

#include <stdbool.h>

#define SIZE 3

// Function to initialize the game board

void initializeBoard(char board[SIZE][SIZE]) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

board[i][j] = '-';

}

}

}

// Function to display the game board

void displayBoard(char board[SIZE][SIZE]) {

printf(" 0 1 2\n");

for (int i = 0; i < SIZE; i++) {

printf("%d ", i);

for (int j = 0; j < SIZE; j++) {

printf("%c ", board[i][j]);

if (j < SIZE - 1) printf("| ");

}

printf("\n");

}

}

// Function to check if a player has won

int checkWin(char board[SIZE][SIZE], char player) {

// Check rows

for (int i = 0; i < SIZE; i++) {

if (board[i][0] == player && board[i][1] == player && board[i][2] == player) {

return 1;

}

}

// Check columns

for (int i = 0; i < SIZE; i++) {

if (board[0][i] == player && board[1][i] == player && board[2][i] == player) {

return 1;

}

}

// Check diagonals

if (board[0][0] == player && board[1][1] == player && board[2][2] == player) {

return 1;

}

if (board[0][2] == player && board[1][1] == player && board[2][0] == player) {

return 1;

}

return 0;

}

// Function to check if the game is a draw

int checkDraw(char board[SIZE][SIZE]) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if (board[i][j] == '-') {

return 0;

}

}

}

return 1;

}

// Function to predict if the game will end in a draw

bool is\_draw\_predicted(char board[SIZE][SIZE], int turn\_count) {

bool x\_can\_win = false;

bool o\_can\_win = false;

if (turn\_count >= 6) {

for (int i = 0; i < SIZE; ++i) {

if (board[i][0] == 'X' && board[i][1] == 'X' && board[i][2] == ' ') x\_can\_win = true;

if (board[i][0] == 'O' && board[i][1] == 'O' && board[i][2] == ' ') o\_can\_win = true;

if (board[0][i] == 'X' && board[1][i] == 'X' && board[2][i] == ' ') x\_can\_win = true;

if (board[0][i] == 'O' && board[1][i] == 'O' && board[2][i] == ' ') o\_can\_win = true;

}

// Check diagonals for X and O

if (board[0][0] == 'X' && board[1][1] == 'X' && board[2][2] == ' ') x\_can\_win = true;

if (board[0][0] == 'O' && board[1][1] == 'O' && board[2][2] == ' ') o\_can\_win = true;

if (board[0][2] == 'X' && board[1][1] == 'X' && board[2][0] == ' ') x\_can\_win = true;

if (board[0][2] == 'O' && board[1][1] == 'O' && board[2][0] == ' ') o\_can\_win = true;

if (!x\_can\_win && !o\_can\_win) {

return true;

}

}

return false;

}

void endGame(char board[SIZE][SIZE], char player) {

if (checkWin(board, player)) {

printf("Player %c wins!\n", player);

} else if (checkDraw(board)) {

printf("It's a draw!\n");

}

}

int main() {

char board[SIZE][SIZE];

char player = 'X';

int row, col;

int turn\_count = 0;

initializeBoard(board);

while (1) {

displayBoard(board);

if (turn\_count >= 6 && is\_draw\_predicted(board, turn\_count)) {

printf("The game is going to end in a draw.\n");

endGame(board, player);

break;

}

printf("Player %c, enter row and column (0-%d): ", player, SIZE - 1);

scanf("%d %d", &row, &col);

if (row < 0 || row >= SIZE || col < 0 || col >= SIZE || board[row][col] != '-') {

printf("Invalid move. Please try again.\n");

continue;

}

board[row][col] = player;

turn\_count++;

if (checkWin(board, player)) {

displayBoard(board);

printf("Player %c wins!\n", player);

break;

}

if (checkDraw(board)) {

displayBoard(board);

printf("It's a draw!\n");

break;

}

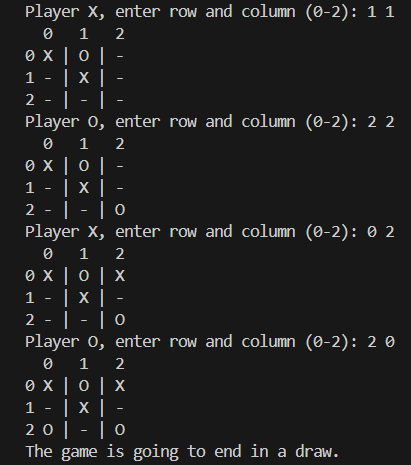
player = (player == 'X') ? 'O' : 'X';

}

return 0;

}

**OUTPUT:**



**2) Implement Tic tac toe using AI**

**CODE:**

// Tic toc toe with AI (MinMax Algo).

#include <stdio.h>

#include <stdbool.h>

#define SIZE 3

#define PLAYER 'X'

#define AI 'O'

#define EMPTY ' '

void printBoard(char board[SIZE][SIZE]) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

printf(" %c ", board[i][j]);

if (j < SIZE - 1) printf("|");

}

printf("\n");

if (i < SIZE - 1) {

printf("---+---+---\n");

}

}

}

bool checkWin(char board[SIZE][SIZE], char player) {

for (int i = 0; i < SIZE; i++) {

if (board[i][0] == player && board[i][1] == player && board[i][2] == player)

return true;

if (board[0][i] == player && board[1][i] == player && board[2][i] == player)

return true;

}

if (board[0][0] == player && board[1][1] == player && board[2][2] == player)

return true;

if (board[0][2] == player && board[1][1] == player && board[2][0] == player)

return true;

return false;

}

bool isBoardFull(char board[SIZE][SIZE]) {

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if (board[i][j] == EMPTY)

return false;

}

}

return true;

}

int evaluate(char board[SIZE][SIZE]) {

if (checkWin(board, AI)) {

return 10;

} else if (checkWin(board, PLAYER)) {

return -10;

} else {

return 0;

}

}

int minimax(char board[SIZE][SIZE], int depth, bool isMaximizing) {

int score = evaluate(board);

if (score == 10 || score == -10) {

return score;

}

if (isBoardFull(board)) {

return 0;

}

if (isMaximizing) {

int best = -1000;

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if (board[i][j] == EMPTY) {

board[i][j] = AI;

int val = minimax(board, depth + 1, !isMaximizing);

best = (val > best) ? val : best;

board[i][j] = EMPTY;

}

}

}

return best;

} else {

int best = 1000;

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if (board[i][j] == EMPTY) {

board[i][j] = PLAYER;

int val = minimax(board, depth + 1, !isMaximizing);

best = (val < best) ? val : best;

board[i][j] = EMPTY;

}

}

}

return best;

}

}

void makeAIMove(char board[SIZE][SIZE]) {

int bestMove = -1;

int bestScore = -1000;

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

if (board[i][j] == EMPTY) {

board[i][j] = AI;

int moveScore = minimax(board, 0, false);

board[i][j] = EMPTY;

if (moveScore > bestScore) {

bestScore = moveScore;

bestMove = i \* SIZE + j;

}

}

}

}

int row = bestMove / SIZE;

int col = bestMove % SIZE;

board[row][col] = AI;

}

void playGame() {

char board[SIZE][SIZE];

for (int i = 0; i < SIZE; i++) {

for (int j = 0; j < SIZE; j++) {

board[i][j] = EMPTY;

}

}

while (true) {

printf("Current board:\n");

printBoard(board);

int row, col;

printf("Player %c's turn. Enter row and column (1-3 for each): ", PLAYER);

scanf("%d %d", &row, &col);

if (row < 1 || row > SIZE || col < 1 || col > SIZE || board[row - 1][col - 1] != EMPTY) {

printf("Invalid move. Please try again.\n");

continue;

}

board[row - 1][col - 1] = PLAYER;

if (checkWin(board, PLAYER)) {

printf("Congratulations! Player %c wins!\n", PLAYER);

printBoard(board);

break;

}

if (isBoardFull(board)) {

printf("It's a draw!\n");

printBoard(board);

break;

}

printf("AI's turn:\n");

makeAIMove(board);

if (checkWin(board, AI)) {

printf("AI wins! Better luck next time.\n");

printBoard(board);

break;

}

if (isBoardFull(board)) {

printf("It's a draw!\n");

printBoard(board);

break;

}

}

}

int main() {

printf("Welcome to 1-Player Tic-Tac-Toe against AI!\n");

playGame();

return 0;

}

**OUTPUT:**

