ASSIGNMENT 7

Perumalla Dharan
AP21110010201

1. **Design an agent to identify the terminal points of a TIC-TAC-TOE game. The agent is supposed to return: [20 points]**
   +1 if agent wins (win for 1)
   0 in case of tie
   -1 if agent loses (win for 0)

```
#include <iostream>
using namespace std;
const int N = 3;

int check_winner(int board[3][3])
{
    for (int i = 0; i < 3; i++)
    {
        int sum = 0;
        for (int j = 0; j < 3; j++)
        {
            sum += board[i][j];
        }
        if (sum == 3)
            return 1;
        if (sum == -3)
            return -1;
    }
    for (int j = 0; j < 3; j++)
    {
        int sum = 0;
```

```
            for (int i = 0; i < 3; i++)
            {
                sum += board[i][j];
            }
            if (sum == 3)
                return 1;
            if (sum == -3)
                return -1;
        }
        int sum1 = board[0][0] + board[1][1] + board[2][2];
        int sum2 = board[0][2] + board[1][1] + board[2][0];

        if (sum1 == 3 || sum2 == 3)
            return 1;
        if (sum1 == -3 || sum2 == -3)
            return -1;

        bool empty = false;
        for (int i = 0; i < 3; i++)
        {
            for (int j = 0; j < 3; j++)
            {
                if (board[i][j] == 0)
                    empty = true;
            }
        }

        if (!empty)
            return 0;

        return -2;
    }

    int main()
```

```
{
    int board[N][N];
    cout<<"Enter the board"<<endl;
    for(int i=0;i<N;i++)
    {
        for(int j=0;j<N;j++)
        {
            cin>>board[i][j];
        }
    }

    int result = check_winner(board);

    if (result == 1)
        cout << "Agent wins" << endl;
    else if (result == -1)
        cout << "Agent loses" << endl;
    else if (result == 0)
        cout << "Tie" << endl;
    else
        cout << "No result yet" << endl;

    return 0;
}
```

```
Enter the board
1 1 -1
-1 -1 1
-1 1 -1
Agent loses
```

```
Enter the board
1 -1 -1
-1 1 1
-1 1 1
Agent wins
```

2. **Extend the agents capability to fill 1 and 0 (user input) alternatively to the vacant positions of the environment of the TIC-TAC-TOE game.**

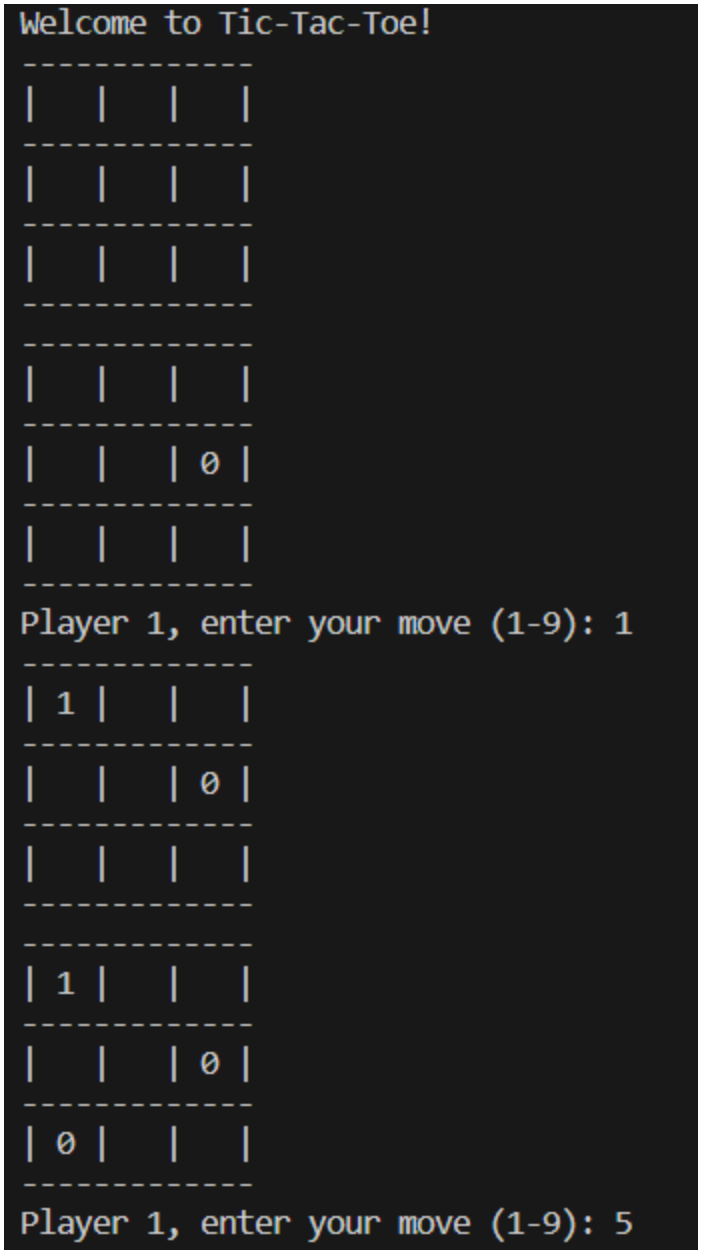
```
#include <iostream>
#include <vector>
#include <ctime>
#include <cstdlib>

using namespace std;

const int BOARD_SIZE = 3;
const char EMPTY = ' ';
const char PLAYER_1 = '1';
const char PLAYER_0 = '0';

vector<vector<char>> board(BOARD_SIZE,
vector<char>(BOARD_SIZE, EMPTY));
void displayBoard()
{
    cout << "-------------" << endl;
    for (int i = 0; i < BOARD_SIZE; ++i)
    {
        cout << "| ";
        for (int j = 0; j < BOARD_SIZE; ++j)
        {
            cout << board[i][j] << " | ";
        }
        cout << endl;
        cout << "-------------" << endl;
    }
}
bool checkWin(char player)
```

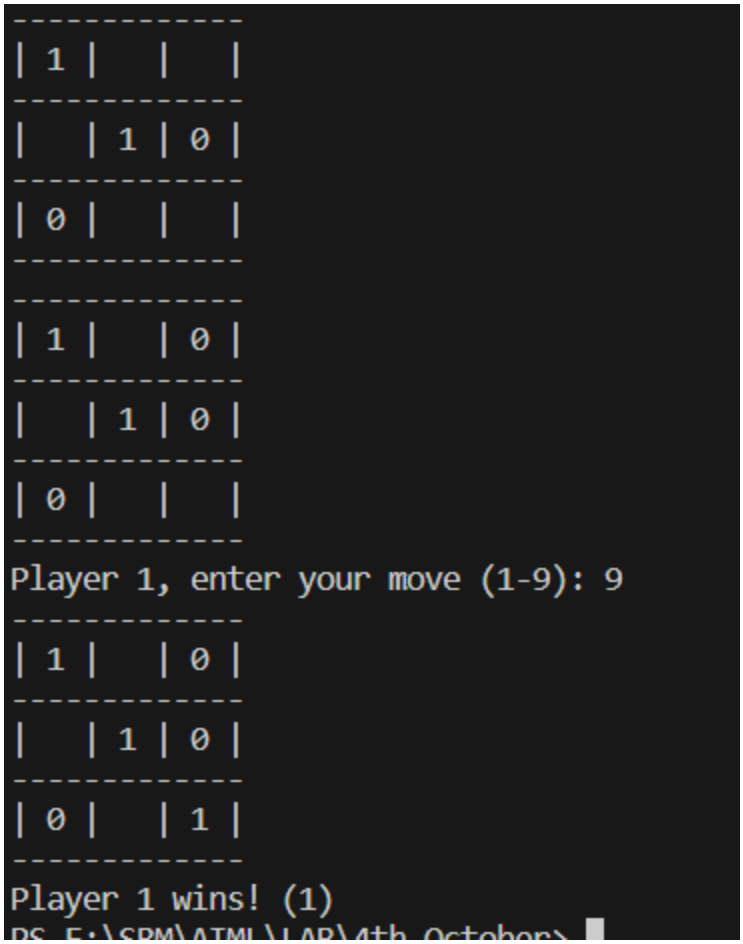
```
{
    for (int i = 0; i < BOARD_SIZE; ++i)
    {
        if (board[i][0] == player && board[i][1] == player
&& board[i][2] == player)
        {
            return true;
        }
    }
    for (int j = 0; j < BOARD_SIZE; ++j)
    {
        if (board[0][j] == player && board[1][j] == player
&& board[2][j] == 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 checkTie()
{
    for (int i = 0; i < BOARD_SIZE; ++i)
    {
```

```
        for (int j = 0; j < BOARD_SIZE; ++j)
        {
            if (board[i][j] == EMPTY)
            {
                return false;
            }
        }
    }
    return true;
}
int determineOutcome()
{
    if (checkWin(PLAYER_1))
    {
        return 1;
    }
    else if (checkWin(PLAYER_0))
    {
        return 0;
    }
    else if (checkTie())
    {
        return -1;
    }
    else
    {
        return -2;
    }
}
void getRandomMove(int &row, int &col)
{
    srand(static_cast<unsigned>(time(nullptr)));
    do
    {
```

```
        row = rand() % BOARD_SIZE;
        col = rand() % BOARD_SIZE;
    } while (board[row][col] != EMPTY);
}

int main()
{
    cout << "Welcome to Tic-Tac-Toe!" << endl;
    displayBoard();

    int currentPlayer = 0;
    while (true)
    {
        int row, col;
        if (currentPlayer == 0)
        {
            getRandomMove(row, col);
        }
        else
        {
            cout << "Player 1, enter your move (1-9): ";
            int move;
            cin >> move;

            if (move < 1 || move > 9)
            {
                cout << "Invalid move! Try again." << endl;
                continue;
            }

            row = (move - 1) / BOARD_SIZE;
            col = (move - 1) % BOARD_SIZE;

            if (board[row][col] != EMPTY)
```

```
            {
                cout << "Invalid move! The cell is already
occupied. Try again." << endl;
                continue;
            }
        }

        board[row][col] = (currentPlayer == 0 ? PLAYER_0 :
PLAYER_1);
        displayBoard();

        int outcome = determineOutcome();
        if (outcome == 1)
        {
            cout << "Player 1 wins! (1)" << endl;
            break;
        }
        else if (outcome == 0)
        {
            cout << "Player 0 wins! (-1)" << endl;
            break;
        }
        else if (outcome == -1)
        {
            cout << "It's a tie! (0)" << endl;
            break;
        }

        currentPlayer = 1 - currentPlayer;
    }

    return 0;
}
```


Welcome to Tic-Tac-Toe!
		0
Player 1, enter your move (1-9): 1
1		
		0
1		
		0
0		
Player 1, enter your move (1-9): 5



3. **Extend the capability of the agent further to select the suitable position for filling 1 (using min max algo).**

```cpp
#include <iostream>
#include <vector>
#include <ctime>
#include <cstdlib>

using namespace std;

const int BOARD_SIZE = 3;
const char EMPTY = '-';
const char PLAYER_1 = 'X';
const char PLAYER_0 = 'O';
```

```
vector<vector<char>> board(BOARD_SIZE,
vector<char>(BOARD_SIZE, EMPTY));

bool checkWin(char player)
{
    for (int i = 0; i < BOARD_SIZE; ++i)
    {
        if (board[i][0] == player && board[i][1] == player
&& board[i][2] == player)
        {
            return true;
        }
    }
    for (int j = 0; j < BOARD_SIZE; ++j)
    {
        if (board[0][j] == player && board[1][j] == player
&& board[2][j] == 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 checkTie()
{
    for (int i = 0; i < BOARD_SIZE; ++i)
    {
        for (int j = 0; j < BOARD_SIZE; ++j)
        {
            if (board[i][j] == EMPTY)
            {
                return false;
            }
        }
    }
    return true;
}

int evaluate()
{
    if (checkWin(PLAYER_1))
    {
        return 1;
    }
    else if (checkWin(PLAYER_0))
    {
        return -1;
    }
    else
    {
        return 0;
    }
}

int minimax(char player)
```

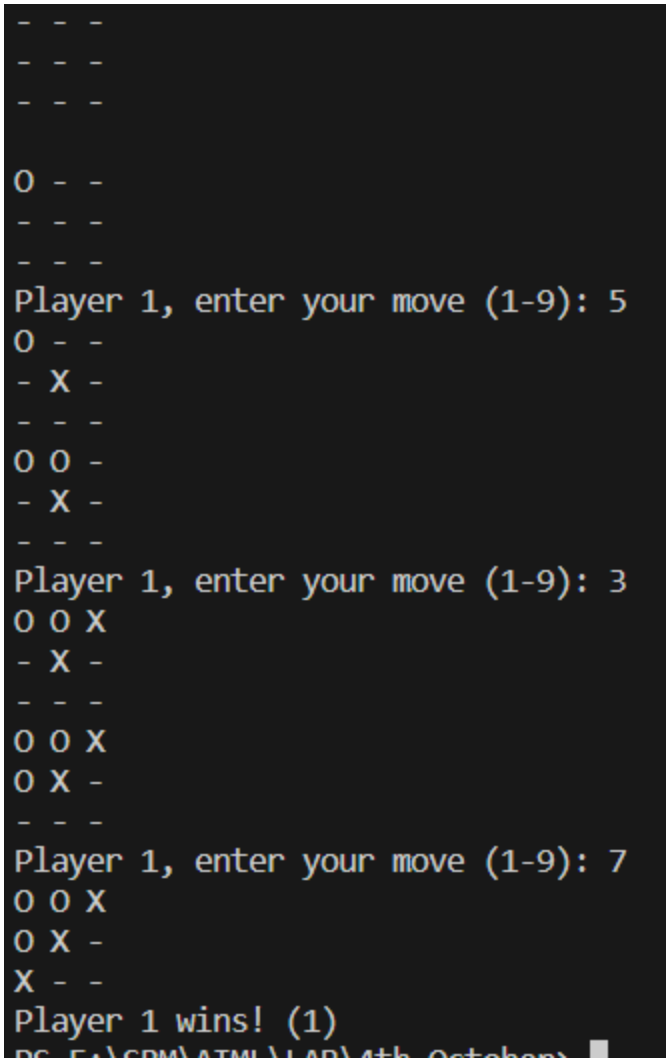
```
{
    int score = evaluate();

    if (score != 0)
    {
        return score;
    }

    int bestScore = (player == PLAYER_1) ? -1000 : 1000;

    for (int i = 0; i < BOARD_SIZE; ++i)
    {
        for (int j = 0; j < BOARD_SIZE; ++j)
        {
            if (board[i][j] == EMPTY)
            {
                board[i][j] = player;
                int currentScore = minimax((player ==
PLAYER_1) ? PLAYER_0 : PLAYER_1);
                board[i][j] = EMPTY;
                if (player == PLAYER_1)
                {
                    bestScore = max(bestScore,
currentScore);
                }
                else
                {
                    bestScore = min(bestScore,
currentScore);
                }
            }
        }
    }
```

```
    return bestScore;
}

void getBestMove(int &row, int &col)
{
    int bestScore = -1000;
    for (int i = 0; i < BOARD_SIZE; ++i)
    {
        for (int j = 0; j < BOARD_SIZE; ++j)
        {
            if (board[i][j] == EMPTY)
            {
                board[i][j] = PLAYER_0;
                int moveScore = minimax(PLAYER_1);
                board[i][j] = EMPTY;
                if (moveScore > bestScore)
                {
                    bestScore = moveScore;
                    row = i;
                    col = j;
                }
            }
        }
    }
}

void displayBoard()
{

    for (int i = 0; i < BOARD_SIZE; ++i)
    {
        for (int j = 0; j < BOARD_SIZE; ++j)
        {
            cout << board[i][j] << " ";
```

```
        }
        cout << endl;
    }
}
int determineOutcome()
{
    if (checkWin(PLAYER_1))
    {
        return 1;
    }
    else if (checkWin(PLAYER_0))
    {
        return 0;
    }
    else if (checkTie())
    {
        return -1;
    }
    else
    {
        return -2;
    }
}

int main()
{
    displayBoard();
    cout << endl;

    int currentPlayer = 0;

    while (true)
    {
        int row, col;
```

```
        if (currentPlayer == 0)
        {
            getBestMove(row, col);
        }
        else
        {
            cout << "Player 1, enter your move (1-9): ";
            int move;
            cin >> move;

            if (move < 1 || move > 9)
            {
                cout << "Invalid move! Try again." << endl;
                continue;
            }

            row = (move - 1) / BOARD_SIZE;
            col = (move - 1) % BOARD_SIZE;

            if (board[row][col] != EMPTY)
            {
                cout << "Invalid move! The cell is already
occupied. Try again." << endl;
                continue;
            }
        }

        board[row][col] = (currentPlayer == 0 ? PLAYER_0 :
PLAYER_1);
        displayBoard();

        int outcome = determineOutcome();
        if (outcome == 1)
        {
```

```
            cout << "Player 1 wins! (1)" << endl;
            break;
        }
        else if (outcome == 0)
        {
            cout << "Player 0 wins! (-1)" << endl;
            break;
        }
        else if (outcome == -1)
        {
            cout << "It's a tie! (0)" << endl;
            break;
        }

        currentPlayer = 1 - currentPlayer;
    }

    return 0;
}
```


- - -
- - -
- - -
O - -
- - -
- - -
Player 1, enter your move (1-9): 5
O - -
- X -
- - -
O O -
- X -
- - -
Player 1, enter your move (1-9): 3
O O X
- X -
- - -
O O X
O X -
- - -
Player 1, enter your move (1-9): 7
O O X
O X -
X - -
Player 1 wins! (1)