VIT Vidyalankar Institute of Technology Accredited A+ by NAAC	DEPARTMENT OF COMPUTER ENGINEERING

Experiment No. 2A

Semester	T.E. Semester VI
Subject	ARTIFICIAL INTELLIGENCE (CSL 604)
Subject Professor In-charge	Prof. Avinash Shrivas
Assisting Teachers	Prof. Avinash Shrivas

Student Name	Deep Salunkhe
Roll Number	21102A0014
Lab Number	310A

Title:

Tic-Tac-Toe implementation

Theory:

Tic-Tac-Toe is a classic game that involves two players, usually denoted as 'X' and 'O', taking turns marking spaces in a 3x3 grid. The objective of the game is to get three of your symbols in a row, column, or diagonal. While Tic-Tac-Toe is a simple game, designing an AI to play it effectively involves implementing various strategies and algorithms.

In your implementation, you've created a simple AI that plays against the user. The AI employs a basic strategy to determine its moves, aiming to both defend against the user's potential winning moves and set up its own winning opportunities.

Ritch and Knight's Algorithm:

The Ritch and Knight's Algorithm is a strategic approach to playing Tic-Tac-Toe, named after its inventors. It involves analyzing the current board state and making moves based on certain priorities. The

algorithm considers blocking the opponent from winning, setting up its own winning moves, and creating opportunities for future wins.

Program Code:

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


void userPlay(vector<int>&board){
    cout<<"1"<<" "<<"2"<<" "<<"3"<<endl;
    cout<<"4"<<" "<<"5"<<" "<<"6"<<endl;
    cout<<"7"<<" "<<"8"<<" "<<"9"<<endl;

    cout<<"Enter the box to write"<<endl;
    int choice;
    cin>>choice;

    if(board[choice-1]==2){
        board[choice-1]=3;

    }else{
        cout<<"Already filled"<<endl;
        userPlay(board);
    }
    return;
}

void computerPlay(vector<int>& board) {
    // Logic for computer playing its turn

    // Define the corners and sides of the board
    vector<int> corners = {0, 2, 6, 8};
    vector<int> sides = {1, 3, 5, 7};

     vector<vector<int>> diagonals = {{0, 4, 8}, {2, 4, 6}};

    // Check if the opponent is about to win and block their winning move
    for (int i = 0; i < 3; ++i) {
        // Check rows
        if ((board[i * 3] == 3 && board[i * 3 + 1] == 3) || (board[i * 3 +1] == 3
&& board[i * 3 + 2] == 3) || (board[i * 3+2] == 3 && board[i * 3 ] == 3) ) {
            if (board[i * 3] == 2) {
                board[i * 3] = 5;
                return;
```

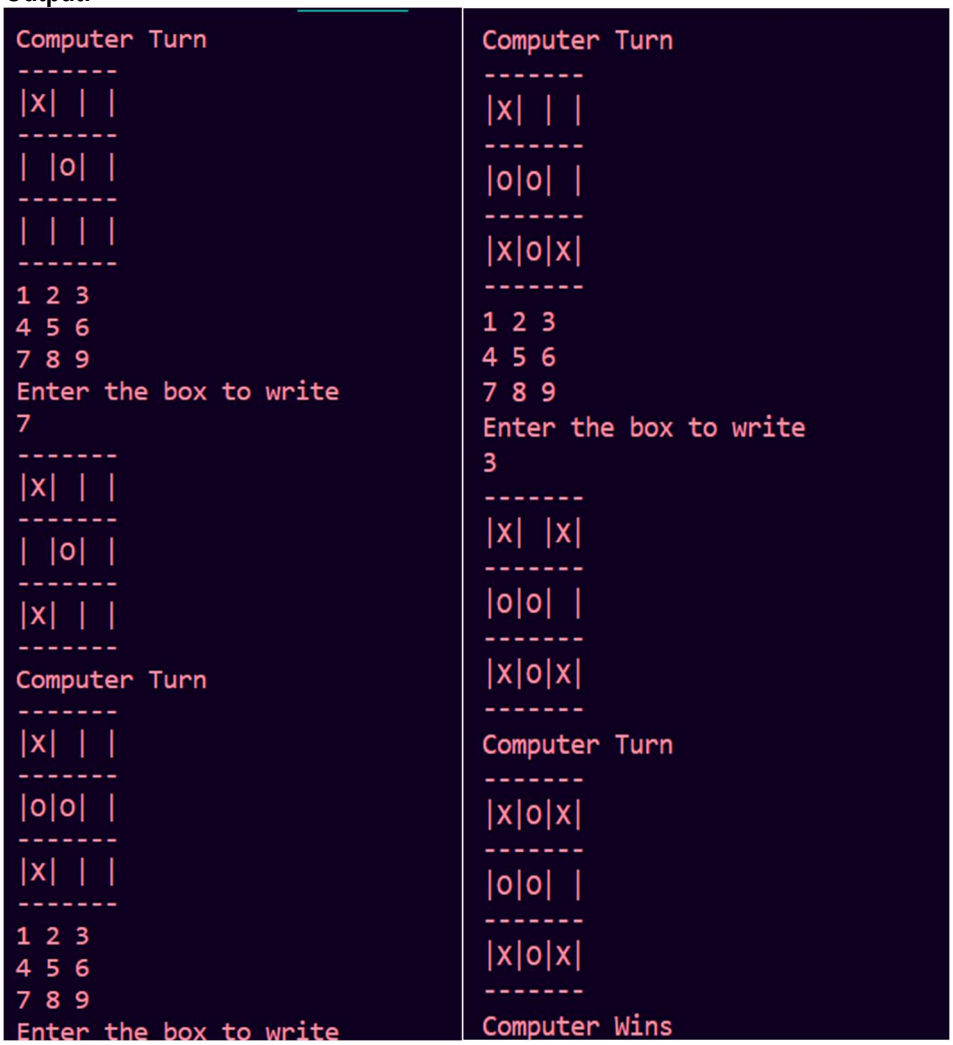
```
            }
            if (board[i * 3 + 1] == 2) {
                board[i * 3 + 1] = 5;
                return;
            }
            if (board[i * 3 + 2] == 2) {
                board[i * 3 + 2] = 5;
                return;
            }
        }
        // Check columns
        if ((board[i] == 3 && board[i + 3] == 3) || (board[i + 6] == 3 && board[i
+ 3] == 3)|| ( board[i] == 3 && board[i + 6] == 3)) {
            if (board[i] == 2) {
                board[i] = 5;
                return;
            }
            if (board[i + 3] == 2) {
                board[i + 3] = 5;
                return;
            }
            if (board[i + 6] == 2) {
                board[i + 6] = 5;
                return;
            }
        }
    }

    // Check diagonals
    for (const auto& diagonal : diagonals) {
        if ((board[diagonal[0]] == 3 && board[diagonal[1]] == 3) || (board[diago-
nal[2]] == 3 && board[diagonal[1]] == 3) || (board[diagonal[0]] == 3 && board[di-
agonal[3]] == 3) ) {
            for (int i : diagonal) {
                if (board[i] == 2) {
                    board[i] = 5;
                    return;
                }
            }
        }
    }

    // If center is available, choose it
    if (board[4] == 2) {
        board[4] = 5;  // Place computer's symbol (e.g., 'O') at the center
    } else {
        // Check if any corner is available and play there
```

```
        for (int i = 0; i < 4; i++) {
            if (board[corners[i]] == 2) {
                board[corners[i]] = 5;  // Place computer's symbol in the corner
                return;
            }
        }

        // If no corner is available, check if any side is available and play
there
        for (int i = 0; i < 4; i++) {
            if (board[sides[i]] == 2) {
                board[sides[i]] = 5;  // Place computer's symbol on the side
                return;
            }
        }
    }
}



void GamePlay(vector<int>&board,int &moveCount){
    if(moveCount%2==1){
        userPlay(board);

    }else{
        cout<<"Computer Turn"<<endl;
        computerPlay(board);

    }
    moveCount++;
    return;
}

void displayBoard(vector<int>&board){
    int index=0;

    for(int i=0;i<3;i++){
        if(i==0)
        cout<<"-------"<<endl;
        for(int j=0;j<3;j++){
            cout<<"|";
            if(board[index]==2){
                cout<<" ";
            }else{
                if(board[index]==3)
                cout<<"X";
                else
```

```
                cout<<"O";
            }
            if(j==2)
            cout<<"|";

            index++;
        }
        cout<<endl;
        cout<<"-------"<<endl;
    }
}

void checkstatus(vector<int>&board,int moveCount){
    //check rows
    for(int i=0;i<3;i++){
        if(board[i*3]==3 && board[i*3+1]==3 && board[i*3+2]==3){
            cout<<"User Wins"<<endl;
            exit(0);
        }
        if(board[i*3]==5 && board[i*3+1]==5 && board[i*3+2]==5){
            cout<<"Computer Wins"<<endl;
            exit(0);
        }
    }

    //check columns
    for(int i=0;i<3;i++){
        if(board[i]==3 && board[i+3]==3 && board[i+6]==3){
            cout<<"User Wins"<<endl;
            exit(0);
        }
        if(board[i]==5 && board[i+3]==5 && board[i+6]==5){
            cout<<"Computer Wins"<<endl;
            exit(0);
        }
    }

    //check diagonals
    if(board[0]==3 && board[4]==3 && board[8]==3){
        cout<<"User Wins"<<endl;
        exit(0);
    }
    if(board[0]==5 && board[4]==5 && board[8]==5){
        cout<<"Computer Wins"<<endl;
        exit(0);
    }
    if(board[2]==3 && board[4]==3 && board[6]==3){
```

```
        cout<<"User Wins"<<endl;
        exit(0);
    }
    if(board[2]==5 && board[4]==5 && board[6]==5){
        cout<<"Computer Wins"<<endl;
        exit(0);
    }


    return;
}


int main(){
    //content of the board
    //2-blank
    //3-x
    //5-y
    vector<int>board(9,2);

    //Move counter
    int moveCount=1;

    //max number of moves in the game
    int game=9;
    cout<<"First Turn for User"<<endl;
    displayBoard(board);
    while(game>=0){

        GamePlay(board,moveCount);

        displayBoard(board);
        if(game==0){
            cout<<"Game Draw"<<endl;
            break;
        }

        checkstatus(board,moveCount);
        game--;
    }

    return 0;

}
```

Output:



Conclusion:

In this lab, we implemented a Tic-Tac-Toe game with a basic AI opponent using the Ritch and Knight's Algorithm. The AI showcases a fundamental understanding of the game's strategy by prioritizing defense and offense to secure a win or prevent a loss. Overall, this lab provided valuable insights into the implementation of AI .