Experiment No 1

Aim: To study and implement Tic Tac Toe problem using AI

Theory:

Game playing is a search problem defined by following components:

1. Initial state: This defines initial configuration of the game and identifies first payer to move.
2. Successor function: This identifies which are the possible states that can be achieved from the current state. This function returns a list of (move, state) pairs, each indicating a legal move and the resulting state.
3. Goal test: Which checks whether a given state is a goal state or not. States where the game ends are called as terminal states.
4. Path cost / utility / payoff function: Which gives a numeric value for the terminal states. In chess, the outcome is win, loss or draw, with values +1, -1, or 0. Some games have wider range of possible outcomes.

Characteristics of game playing

1. Unpredictable Opponent: Generally we cannot predict the behavior of the opponent. Thus we need to find a solution which is a strategy specifying a move for every possible opponent move or every possible state.
2. Time Constraints: Every game has a time constraints. Thus it may be infeasible to find the best move in this time.

How to Play a Game in AI?

Typical structure of the game in the AI is:

- 2- person game
- Players alternate moves
- Zero-sum game: one player’s loss is the other’s gain
- Perfect information: both players have access to complete information about the state of the game. No information is hidden from either player.
- No chance (e. g. using dice) involved

E.g. Tic- Tac- Toe, Checkers, Chess, Go, Nim, Othello

For a Tic Tac Toe Problem.

Tic-tac-toe, also called noughts and crosses (in the British Commonwealth countries) and **X's and O's** in the Republic of Ireland, is a pencil-and-paper game for two players, **X** and **O**, who take turns marking the spaces in a 3×3 grid. The **X** player usually goes first. The player who succeeds in placing three respective marks in a horizontal, vertical, or diagonal row wins the game.

1. Initial state: Board Configuration at start. Usually empty
2. Successor function: This identifies which are the possible states that can be achieved from the current state. This function returns a list of (move, state) pairs, each indicating a legal move and the resulting state.
3. Goal test: Which checks whether a given state is a goal state or not. If there are 3 **X's or O's** in a row, horizontally, vertically or diagonally.
4. Path cost / utility / payoff function: Which gives a numeric value for the terminal states. The outcome is win, loss or draw, with values +1, -1, or 0.

Code:

```
#include<stdio.h>
int winner(int);
int enter(int,int);
int cur(void);
int pos,counter=0;
int a[3][3];
int cur(){
int i,j;
//showing current state
printf("\nGame board after entering your turn!:");
for(i=0;i<3;i++){
printf("\n");
for(j=0;j<3;j++){
printf("%d ",a[i][j]);}}
return 0;}
int main(){
int b[3][3],i,j,k=0,z;
printf("\t\t\tTIC TAC TOE\n");//DISPLAYING POSITIONS
for(i=0;i<3;i++)
```

```
for(j=0;j<3;j++){
b[i][j]=++k;}
printf("These are the positions:\n");
for(i=0;i<3;i++){
printf("\n");
for(j=0;j<3;j++){
printf("%d ",b[i][j]);}}
//initializing matrix
for(i=0;i<3;i++)
for(j=0;j<3;j++){
a[i][j]=5;}//ASKING FOR INPUT
for(z=1;z<=9;z++){//Player 1
if(z%2!=0){
printf("\nPlayer 1 enter your choice of position:");
scanf("%d",&pos);
enter(1,pos);
cur();
winner(pos);
if(counter>0){
break;}}
else{
printf("\nPlayer 2 enter your choice of position:");
scanf("%d",&pos);
enter(2,pos);
cur();
winner(pos);
if(counter>0){
break;}}}
if(counter==0){
printf("Match is draw!");}}
//ENTERING VALUES
int enter(int p,int pos){
if(pos==1){
a[0][0]=p;}
if(pos==2){
a[0][1]=p;}
if(pos==3){
a[0][2]=p;}
if(pos==4){
a[1][0]=p;}
if(pos==5){
a[1][1]=p;}
if(pos==6){
a[1][2]=p;}
if(pos==7){
a[2][0]=p;}
```

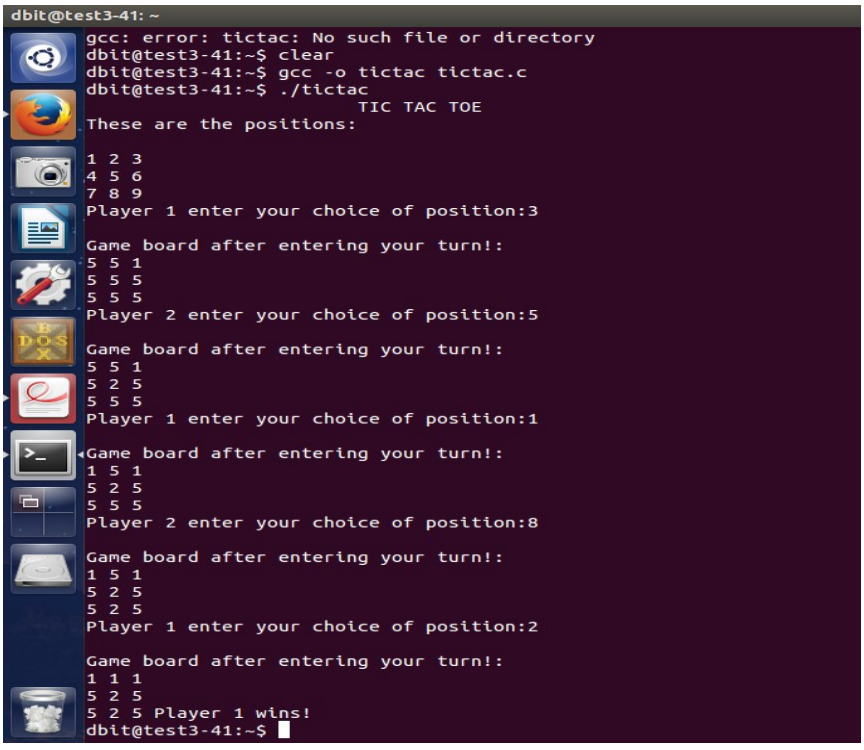
```
if(pos==8){
a[2][1]=p;}
if(pos==9){
a[2][2]=p;
}return 0;}//finding out winner
int winner(int pos){
if(pos == 1){
//Horizontal
if (a[0][0] == a[0][1] && a[0][0] == a[0][2]){
if(a[0][0] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}//Vertical
if(a[0][0] == a[1][0] && a[0][0] == a[2][0]){
if(a[0][0] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//diagonal
if(a[0][0] == a[1][1] && a[0][0] == a[2][2]){
if(a[0][0] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}}
if(pos == 2){
//Horizontal
if (a[0][0] == a[0][1] && a[0][0] == a[0][2]){
if(a[0][1] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//Vertical
if(a[0][1] == a[1][1] && a[0][1] == a[2][1])
{if(a[0][1] == 1){
printf("Player 1 wins!\n");
counter++;}else{
printf("Player 2 wins!\n");
counter++;}}}
```

```
if(pos == 3){
//Horizontal
if (a[0][0] == a[0][1] && a[0][0] == a[0][2]){
if(a[0][2] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//Vertical
if(a[0][2] == a[1][2] && a[0][2] == a[2][2]){
if(a[0][2] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//diagonal
if(a[0][2] == a[1][1] && a[0][2] == a[2][0]){
if(a[0][2] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}}
if(pos == 4){
//Horizontal
if (a[1][0] == a[1][1] && a[1][0] == a[1][2]){
if(a[1][0] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}//Vertical
if(a[0][0] == a[1][0] && a[1][0] == a[2][0]){
if(a[1][0] == 1){
printf("Player 1 wins!\n");counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}}
if(pos == 5){
//Horizontal
if (a[1][0] == a[1][1] && a[1][0] == a[1][2]){
if(a[1][1] == 1)
{printf("Player 1 wins!\n");
counter++;}
```

```
else{
printf("Player 2 wins!\n");
counter++;}}
//Vertical
if(a[0][1] == a[1][1] && a[1][1] == a[2][1]){
if(a[1][1] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//diagonal
if(a[0][0] == a[1][1] && a[1][1] == a[2][2]){
if(a[0][0] == 1)
{printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//diagonal 2
if(a[0][2] == a[1][1] && a[1][1] == a[2][0]){
if(a[0][0] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}}
if(pos == 6){
//Horizontal
if (a[1][0] == a[1][2] && a[1][2] == a[1][1]){if(a[1][2] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//Vertical
if(a[0][2] == a[1][2] && a[1][2] == a[2][2]){
if(a[1][2] == 1){
printf("Player 1 wins!\n");
counter++;
}else{
printf("Player 2 wins!\n");
counter++;}}}
if(pos == 7){
//Horizontal
if (a[2][0] == a[2][1] && a[2][0] == a[2][2]){
```

```
if(a[2][0] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//Vertical
if(a[0][0] == a[1][0] && a[0][0] == a[2][0]){
if(a[2][0] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//diagonal
if(a[2][0] == a[1][1] && a[2][0] == a[0][2]){
if(a[2][0] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}}
if(pos == 8){//Horizontal
if (a[2][0] == a[2][1] && a[2][0] == a[2][2]){
if(a[2][1] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//Vertical
if(a[0][1] == a[1][1] && a[0][1] == a[2][1]){
if(a[2][1] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;
}}}
if(pos == 9){
//Horizontal
if (a[2][0] == a[2][1] && a[2][0] == a[2][2]){
if(a[0][2] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
```

```
printf("Player 2 wins!\n");
counter++;
}}
//Vertical
if(a[0][2] == a[1][2] && a[0][2] == a[2][2]){
if(a[2][2] == 1){
printf("Player 1 wins!\n");
counter++;}
else{
printf("Player 2 wins!\n");
counter++;}}
//diagonal
if(a[2][2] == a[1][1] && a[2][2] == a[0][0]){
if(a[0][2] == 1){
printf("Player 1 wins!\n");
counter++;}else{
printf("Player 2 wins!\n");
counter++;
}
}
}
```



Conclusion: We have implemented the TicTacToe game using Artificial Intelligence by using the concept of search technique.