ASSIGNMENT NO: 1

AIM: Implement Tic-Tac-Toe game

PROBLEM STATEMENT: Implement Tic-Tac-Toe using A* algorithm

PREREQUISITES: Min Max Algorithm

COURSE OBJECTIVE:

- To understand the use of searching in artificial intelligence
- To understand the optimal searching techniques
- To understand the concept of A* algorithm

COURSE OUTCOMES:

Students will be able to,

Implement Tic-Tac-Toe game using A* algorithm

ALGORITHM / PSEUDOCODE:

Tic-Tac-Toe – Finding optimal move **1.Finding the Best Move Pseudocode :**

```
: function findBestMove(board):
bestMove = NULL
    for each move in board :
        if current move is better than
            bestMove bestMove = current move
return bestMove
```

2. Minimax Pseudocode:

```
function minimax(board, depth, isMaximizingPlayer):
    if current board state is a terminal state
                : return value of the board


if isMaximizingPlayer : bestVal = -
INFINITY  for each move in board :
value = minimax(board, depth+1,
false)  bestVal = max( bestVal, value)
return bestVal

else :
bestVal = +INFINITY  for each move
in board :  value = minimax(board,
depth+1, true)  bestVal = min(
bestVal, value)  return bestVal
```

3.Checking for GameOver

```
 state function
 isMovesLeft(board):
   for each cell in board:
     if current cell is
       empty: return true
return false
```

CODE:

```
package Air_Prasad;

import java.util.Scanner;

public class TicTacToe {

             static Scanner in = new Scanner(System.in);

             static String board[][] = new String[3][3];

             static int moveCount;

             static String humanPlayer;                    //holds symbol (X|O)

             static String aiPlayer;

  static int moveI,moveJ;    //board indexes of current AI moves - used in printBoard()

             public static void init(){
      //Initializing board


                    for (int i=0;i<3;i++)
                          for(int j=0;j<3;j++){
                                 board[i][j] = " ";
                          }
             }

             public static void selectSymbol(){
      //Select Symbol (X | O)

                    String input = "";

                    System.out.println("Choose X|O : ");
                    input = in.next();

                    while(!input.equalsIgnoreCase("x") && !input.equalsIgnoreCase("o")){
                          System.out.println("\nInvalid Input! (Only X or O)\n");
                          System.out.println("Choose X|O : ");
                          input = in.next();
                    }

                    humanPlayer = (input.equalsIgnoreCase("x")) ? "X" : "O";   aiPlayer =
(humanPlayer.equals("X")) ? "O" : "X";


}
```

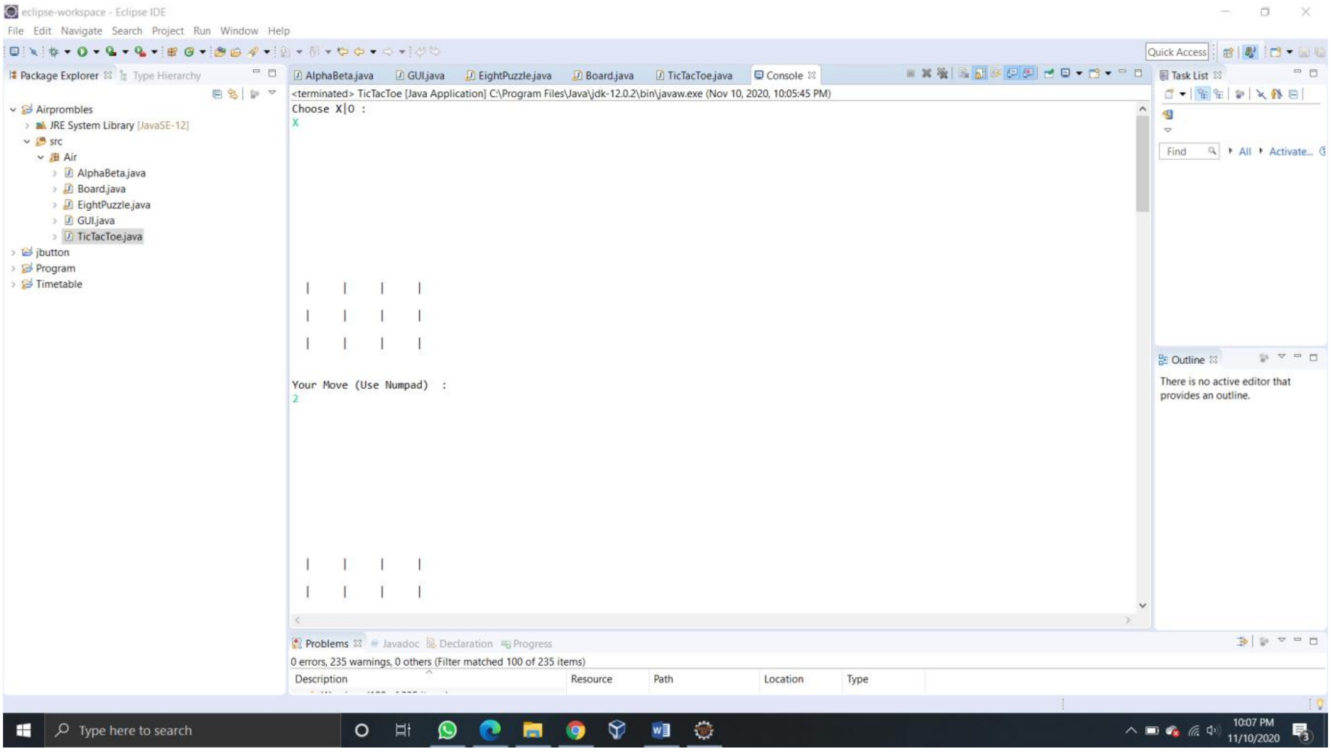
```
public static void printBoard(){
System.out.println("\n\n\n\n\n\n\n\n\n\n" ); for (int i=0;i<3;i++){
for(int j=0;j<3;j++){

                    System.out.print("  |  "+board[i][j]);
                                        }
                          System.out.println("  |\n");
                          //System.out.println("\n\t\t--\t\t--\t\t--\n");
                    }
                    System.out.println();




                    if(checkWin(board)=="nothing")
                    {
                          if(moveCount!=0)
                          {
                                 System.out.println("AI Player marked "+aiPlayer+" at
position "+(moveI+1)+"-"+(moveJ+1));
                                 System.out.println("Your Move (Use Numpad)    : ");
                          }
                          else
                          {
                                 System.out.println("Your Move (Use Numpad)    : ");
                          }
                    }
             }

  public static boolean validMove(int boardNumber)  //Check if position on board is empty
             {
                    int indexes[] = getBoardIndexes(boardNumber);


                    if(!board[indexes[0]][indexes[1]].equals(" "))
      return false;
                    return true;

             }

             public static void inputMove(){
      //input move from user (1-9) - Numpad

                    int humanPlayerInput = 0;


                    do{
                          try{
                    humanPlayerInput = in.nextInt();
                    if((humanPlayerInput < 1 || humanPlayerInput > 9))
                    System.out.println("Invalid Input (Only 1-9)");      else
if(!validMove(humanPlayerInput))
                    System.out.println("Invalid Move! Place Already Marked!");

             }catch(Exception e)
             {
                    System.out.println("Invalid Input (Only 1-9)");
             in.next();
             }
}
```

```
while(humanPlayerInput < 1 || humanPlayerInput > 9 ||
!validMove(humanPlayerInput));

                    int indexes[] = getBoardIndexes(humanPlayerInput);
                    board[indexes[0]][indexes[1]] = humanPlayer;

             }

             public static int[] getBoardIndexes(int boardNumber)   //Convert 1-9 to board
indexes
             {
                    int []indexes = new int[2];


                    switch(boardNumber){

                    case 7:
                           indexes[0] = 0;                         indexes[1] = 0;
                    return indexes;
                                               case 8:
                           indexes[0] = 0;                         indexes[1] = 1;
                           return indexes;

                    case 9:
                           indexes[0] = 0;                         indexes[1] = 2;
                           return indexes;

                    case 4:
                           indexes[0] = 1;                         indexes[1] = 0;
                    return indexes;                  case 5:
                           indexes[0] = 1;                         indexes[1] = 1;
                    return indexes;                  case 6:
                           indexes[0] = 1;                         indexes[1] = 2;
                    return indexes;                  case 1:
                           indexes[0] = 2;                         indexes[1] = 0;
             return indexes;     case 2:
             indexes[0] = 2;     indexes[1] = 1;     return indexes; case 3:
       indexes[0] = 2;     indexes[1] = 2;     return indexes; }

return indexes;


             }
             public static boolean isHumanTurn(){                    //Check if its human's

                    if(moveCount % 2 == 0)
                           return true;
                    return false;
             }

public static void findBestMove(){                                 //Find best move

                    int bestValue = -10000;
                     moveI = -1;
                     moveJ = -1;

                    for(int i=0;i<3;i++)
                           for(int j=0;j<3;j++)
                                 if(board[i][j] == " ")
                                 {
```

```
                        board[i][j] = aiPlayer; //make move

                        int val = getBoardValue(board,0,false);
    //send new board to getBoardValue

                        board[i][j] = " "; //undo move

                        if(val > bestValue)
                        {

                            bestValue = val;

                            moveI = i;
                            moveJ = j;

                        }
                    }

                board[moveI][moveJ] = aiPlayer;


        }

        public static int getBoardValue(String newBoard[][],int depth,boolean isMax){
//Calculates values of every possible board


if(checkWin(newBoard) == "draw")
        return 0 + depth;        //draw in max possible moves

if(checkWin(newBoard) == aiPlayer)
    return 10 - depth;        // win in min possible moves

        if(checkWin(newBoard) == humanPlayer)
    return -10 + depth;    // lose in max possible moves


            if(isMax)  //if score is to be maximized : in case of AI Player
            {
                int maxValue = -100;

                for(int i=0;i<3;i++)
                    for(int j=0;j<3;j++)
                        if(newBoard[i][j] == " "){

                            newBoard[i][j] = aiPlayer;  //make move

                            int value = getBoardValue(newBoard,
depth+1,!isMax);

                            newBoard[i][j] = " ";            //undo move

                            if(value > maxValue)
                                maxValue = value;

                        }

                return maxValue;

            }
            else   //if score is to be minimized : in case of Human Player
            {
                int minValue = 10000;

                for(int i=0;i<3;i++)
                    for(int j=0;j<3;j++)
                        if(newBoard[i][j] == " "){

                            newBoard[i][j] = humanPlayer;  //make move

                        int value = getBoardValue(newBoard, 
depth+1,!isMax);
                            newBoard[i][j] = " ";           //undo move

                            if(value < minValue)
                                minValue = value;

                        }

                return minValue;

            }


        }

        public static String checkWin(String newBoard[][]){  //Accepts a board and 
Checks if User/AI has won, or draw
//returns win or draw or nothing

for(int i=0;i<3;i++)  //checking rows & cols
{
        if(newBoard[i][0].equals(newBoard[i][1]) && 
newBoard[i][1].equals(newBoard[i][2]))
                {
                    if(!newBoard[i][0].equals(" "))
                    if(newBoard[i][0] == humanPlayer)
                        return humanPlayer;
                    else
                        return aiPlayer;
                }

                if(newBoard[0][i].equals(newBoard[1][i]) && 
newBoard[1][i].equals(newBoard[2][i]))
                {
                    if(!newBoard[0][i].equals(" "))
                    if(newBoard[0][i] == humanPlayer)
                        return humanPlayer;
                    else
                        return aiPlayer;
                }
            }

            if((newBoard[0][0].equals(newBoard[1][1]) && 
newBoard[1][1].equals(newBoard[2][2])) || (newBoard[0][2].equals(newBoard[1][1]) && 
newBoard[1][1].equals(newBoard[2][0])))
            {
                if(!newBoard[1][1].equals(" "))
                if(newBoard[1][1] == humanPlayer)
    return humanPlayer;
                else return aiPlayer;
            }
```

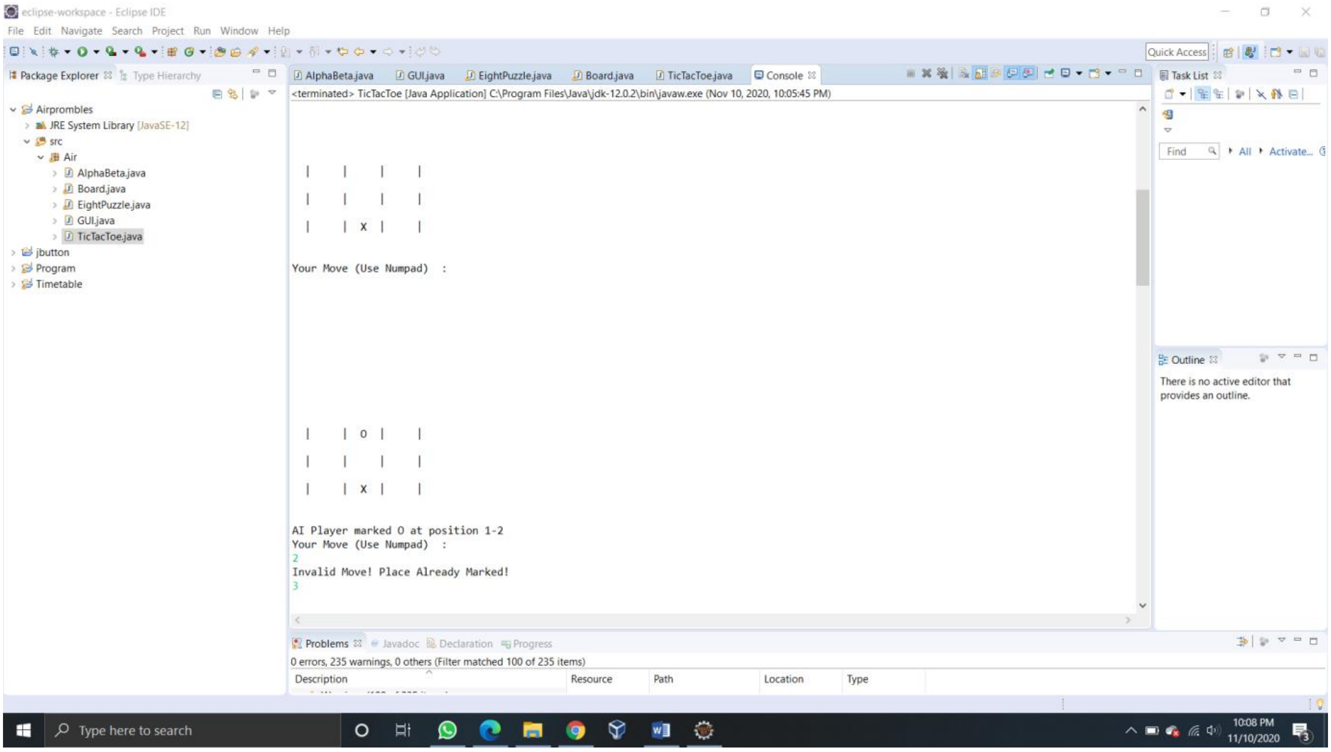
```
   for(int i=0;i<3;i++)     for(int j=0;j<3;j++)
                                 if(newBoard[i][j] == " ")
                                        return "nothing";


                        return "draw";

                }

  public static void aiMove(){      //calls findBestMove()

                        findBestMove();

                }

                public static void play(){                                       //keep
playing while value returned from checkWin() is "nothing"

                        moveCount = 0;
printBoard();

while(checkWin(board) == "nothing")
{

                if(isHumanTurn())
                                        inputMove();
                                else
                                        aiMove();

                                printBoard();
                                moveCount++;
                        }

   if(checkWin(board)==humanPlayer)     System.out.println("You win!");    else
if(checkWin(board) == aiPlayer)     System.out.println("AI Player wins!");    else
System.out.println("Draw!");

                }

                public static void main(String args[])
                {

                        init();
                        selectSymbol();                  play();

                }
}
```

Output:



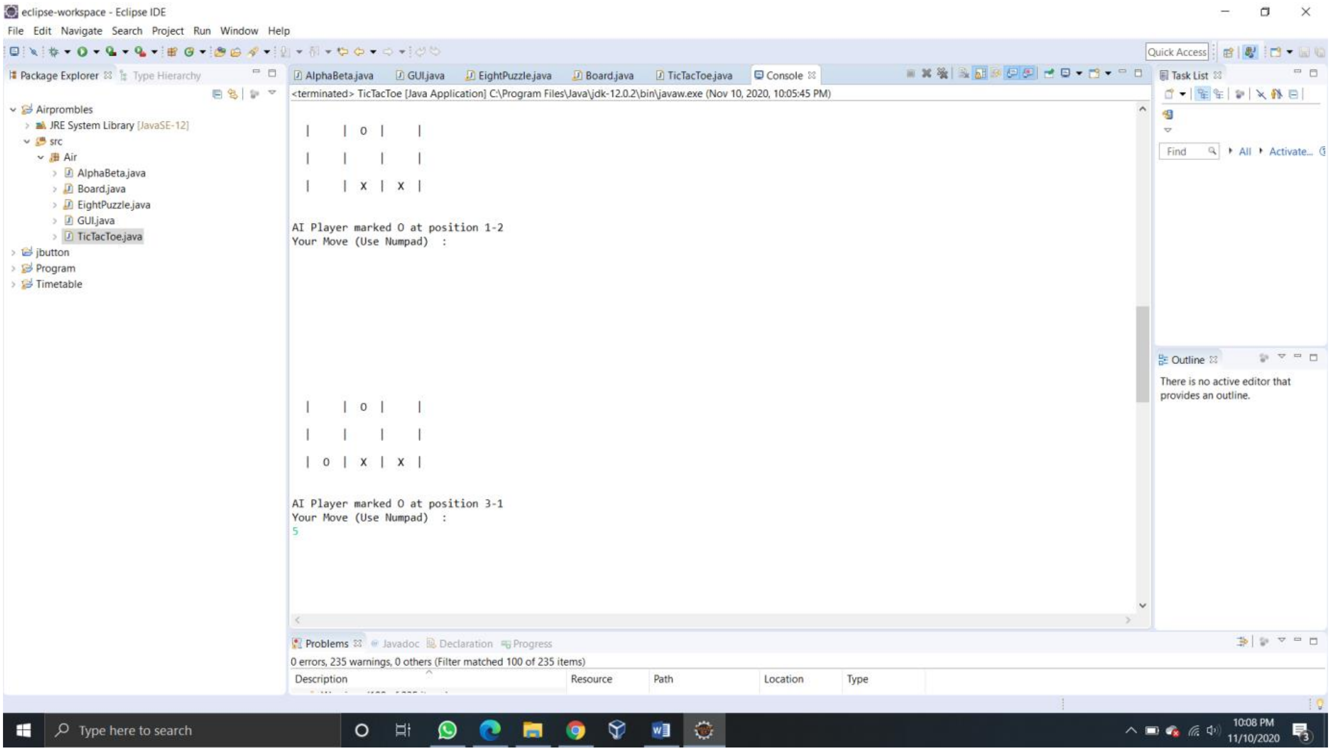
```
<terminated> TicTacToe [Java Application] C:\Program Files\Java\jdk-12.0.2\bin\javaw.exe (Nov 10, 2020, 10:05:45 PM)
Choose X|O :
X









|   |   |   |

|   |   |   |

|   |   |   |


Your Move (Use Numpad)  :
2











|   |   |   |

|   |   |   |
```

0 errors, 235 warnings, 0 others (Filter matched 100 of 235 items)



```
<terminated> TicTacToe [Java Application] C:\Program Files\Java\jdk-12.0.2\bin\javaw.exe (Nov 10, 2020, 10:05:45 PM)



|   |   |   |

|   |   |   |

|   | X |   |


Your Move (Use Numpad)  :










|   | O |   |

|   |   |   |

|   | X |   |


AI Player marked O at position 1-2
Your Move (Use Numpad)  :
2
Invalid Move! Place Already Marked!
3
```

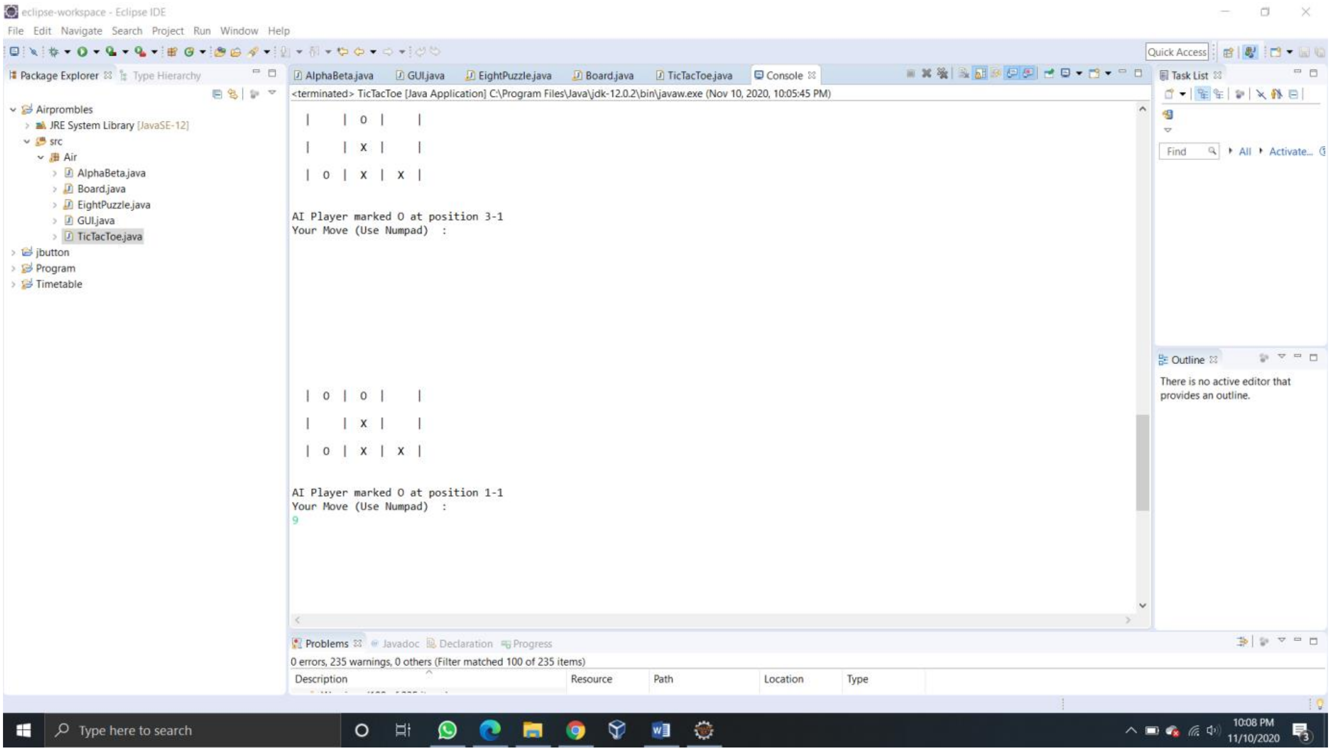
0 errors, 235 warnings, 0 others (Filter matched 100 of 235 items)



```
<terminated> TicTacToe [Java Application] C:\Program Files\Java\jdk-12.0.2\bin\javaw.exe (Nov 10, 2020, 10:05:45 PM)

 |   | O |   |

 |   |   |   |

 |   | X | X |


AI Player marked O at position 1-2
Your Move (Use Numpad)  :




 |   | O |   |

 |   |   |   |

 | O | X | X |


AI Player marked O at position 3-1
Your Move (Use Numpad)  :
5
```

0 errors, 235 warnings, 0 others (Filter matched 100 of 235 items)

Description	Resource	Path	Location	Type



```
<terminated> TicTacToe [Java Application] C:\Program Files\Java\jdk-12.0.2\bin\javaw.exe (Nov 10, 2020, 10:05:45 PM)
 |   | O |   |

 |   | X |   |

 | O | X | X |


AI Player marked O at position 3-1
Your Move (Use Numpad)  :




 | O | O |   |

 |   | X |   |

 | O | X | X |


AI Player marked O at position 1-1
Your Move (Use Numpad)  :
9
```

0 errors, 235 warnings, 0 others (Filter matched 100 of 235 items)

Description	Resource	Path	Location	Type

CONCLUSION: In this assignment we have successfully implemented the tic-tac-toe using A* algorithm.

FAQS:

1. **What do mean by zero sum game?**

Ans: Zero-sum is a situation in game theory in which one person’s gain is equivalent to another’s loss, so the net change in wealth or benefit is zero. A zero-sum game may have as few as two players or as many as millions of participants.

2. **What is the difference between deterministic and stochastic games?** Ans: In deterministic games, the output of the game mpdel is fully determined by the parameter values and the initial conditions initial conditions. Stochastic game models possess some inherent randomness. The same set of parameter values and initial conditions will lead to an ensemble of different outputs.

3. **How the A* algorithm works?**

Ans: A* works by making a lowest-cost path tree from the start node to the target node. What makes A* different and better for many searches is that for each node, A* uses a function f(n)f(n) that gives an estimate of the total cost of a path using that node

ASSIGNMENT NO: 2

AIM: Write a program to implement Eight-Puzzle problem.

PROBLEM STATEMENT: Solve 8-puzzle problem using A* algorithm. Assume any initial configuration and define goal configuration clearly.

COURSE OBJECTIVES:

- Understand and analyze the problem definition of 8- puzzle problem. Implement eight puzzle problem.

COURSE OUTCOMES:

Students will be able to,

- Analyze local and global impact of computing for specific problem using 8-puzzle problem.

- To formulate and solve 8-puzzle problem

Analyze current techniques and tools for solving 8-puzzle

ALGORITHM/PSEUDOCODE:

```
function A*(start,goal) closedset := the empty set       // The set of
nodes already evaluated.
 openset := set containing the initial node evaluated. came_from := the empty
                map     // The map of navigated nodes.
                g_score[start] := 0                  // Distance from start along optimal path.
        h_score[start] := heuristic_estimate_of_distance(start, goal) f_score[start] :=
                h_score[start] // Estimated total distance from start to goal
through y. while openset is not empty
                x := the node in openset having the lowest f_score[]
                value if x = goal
        return reconstruct_path(came_from, came_from[goal])
                remove x from openset
                add x to closedset
        foreach y in
                neighbor_nodes(x) if y in
                closedset continue
        tentative_g_score := g_score[x] + dist_between(x,y)

                if y not in openset add y to openset
                tentative_is_better := true elseif
                tentative_g_score < g_score[y]
                tentative_is_better := true else
                tentative_is_better := false if
                tentative_is_better  =  true
                came_from[y] := x
                g_score[y]  :=  tentative_g_score  h_score[y]  :=
        heuristic_estimate_of_distance(y,  goal)  f_score[y]
        := g_score[y] + h_score[y] return failure
        function reconstruct_path(came_from, current_node)
             if came_from[current_node] is set
             p = reconstruct_path(came_from, came_from[current_node])
        return (p + current_node) else
return current_node
```

Code:

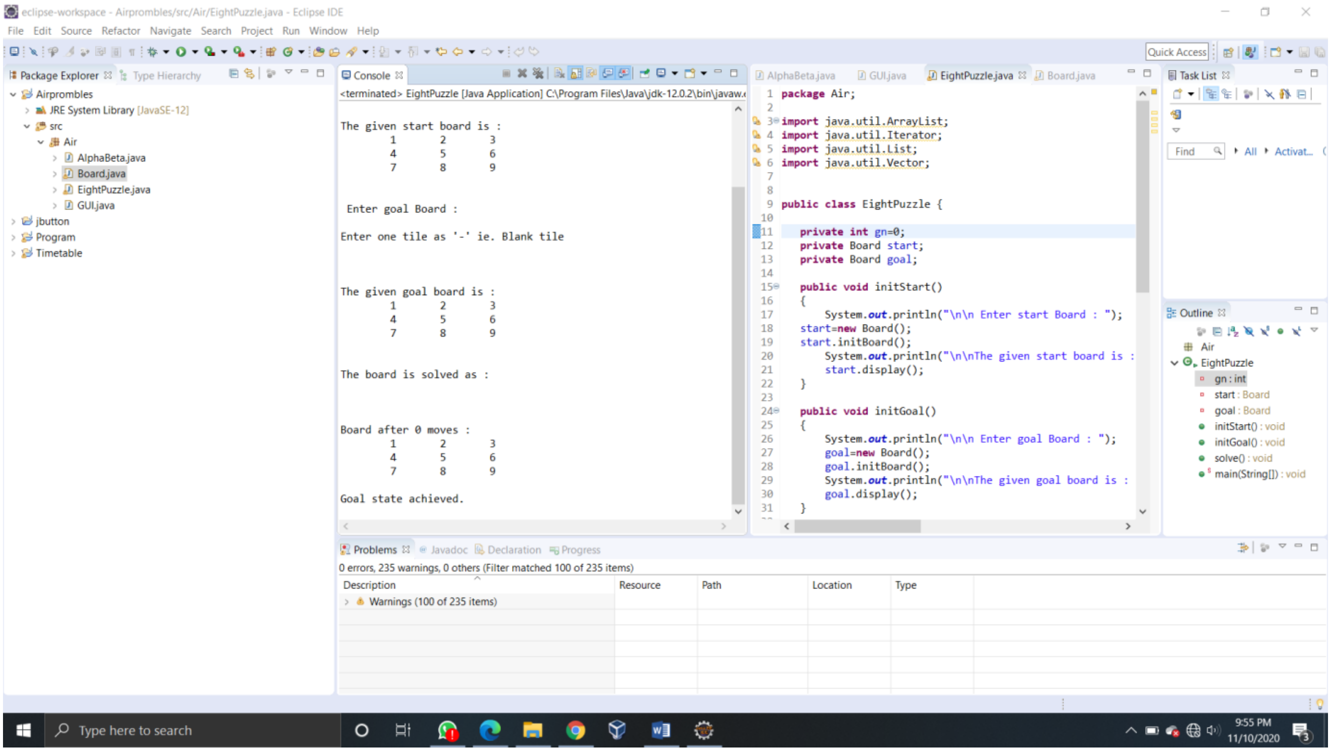
```
package Air_Prasad;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;



public class EightPuzzle {

  private int gn=0;                                  // Initialize gn ie. no. of moves to 0
  private Board start;
  private Board goal;

  public void initStart()                             //Accept and display start board
  {
     System.out.println("\n\n Enter start Board : ");
  start=new Board();
  start.initBoard();
     System.out.println("\n\nThe given start board is : ");
     start.display();
  }

  public void initGoal()                              //Accept and display goal board
  {
     System.out.println("\n\n Enter goal Board : ");
     goal=new Board();
     goal.initBoard();
     System.out.println("\n\nThe given goal board is : ");
     goal.display();
  }
  public void solve()                              // Solve puzzle using A* algorithm
  {
     Board cur = start;
     while(true)
     {
        System.out.println("\n\nBoard after "+gn+" moves : ");
        cur.display();
        if(cur.equals(goal))                       //Check if goal is achieved nad return
        {
           System.out.println("\nGoal state achieved.");
           return;
        }

        gn++;                                      // Increment gn as per moves



        cur = cur.nextMove(gn, goal);              // get the board after next move
     }
  }

  public static void main(String[] args) {

     EightPuzzle ep = new EightPuzzle();           // Instantiate and solve the puzzle
     ep.initStart();
     ep.initGoal();

     System.out.println("\n\nThe board is solved as : \n");
     ep.solve();




  } }
```

Class to Board:-

```
package Air_Prasad;

import java.util.Scanner;
import javax.swing.JOptionPane;
```

```
//board class for eight puzzle matrix
public class Board {

    private String board[][];
    private  int blankX,blankY;                                                       // co-
ordinates for blank tile

    public Board()
    {
        this.board = new String[3][3];
    }

    public Board(Board b)
//constructor to initialise Board
    {
        this.board = b.board;         this.blankX = b.blankX;         this.blankY =
b.blankY;
    }
    public void initBoard()
//initialize the board
    {
        Scanner inp = new Scanner(System.in);
        System.out.println("\nEnter one tile as '-' ie. Blank tile\n");         for(int
i=0; i<3; i++)
        {
            for(int j=0; j<3; j++)
            {
                board[i][j] = JOptionPane.showInputDialog("Enter the value of tile
["+i+"]["+(j)+"] : ");

                if(board[i][j].equals("-"))
//store the location of blank symbol
                {
                    blankX=i;                       blankY=j;
                }
            }
        }
    }

    public String[][] getBoard()
    {
        return board;
    }

    public void setBoard(String[][] board)                                            // Set
the board puzzle matrix
    {
        for(int i=0; i<3; i++)
        {
            for(int j=0; j<3; j++)
            {
                this.board[i][j] = board[i][j];
            }
        }
    }



    public int getBlankX()
    {
        return blankX;
    }

     public int getBlankY()
    {
        return blankY;
    }

     public void setBlankX(int x)
    {
        blankX = x;
    }

     public void setBlankY(int y)
    {
        blankY = y;
    }

    public void display()
    {
        for(int i=0; i<3; i++)
        {
            for(int j=0; j<3; j++)
            {
                System.out.print("\t"+board[i][j]);
            }
            System.out.println();
        }
    }

    public Board nextMove(int gn, Board goal)                              //method to
check possible moves and select optimum
    {
        Board temp = new Board();         Board next = new Board();
        int minFn = 999;
        System.out.println("\nPossible moves are : ");
        if(blankY>0)                                                      // Condition for
possible left move
        {
            temp.setBoard(board);
            temp.swap(blankX, blankY, blankX, blankY-1);                  // Swap blank
tile
            int fn =  (temp.getHn(goal)+gn);                              // Calculate fn
= hn + gn
            System.out.println("\nFor Fn = "+fn+" : ");
            temp.display();
            if(fn < minFn)                                                // Check for
minimum fn and set the next board accordingly
            {
                minFn = fn;
                next.setBoard(temp.board);
                next.setBlankX(blankX);
                next.setBlankY(blankY-1);
            }
                     }
        if(blankY<2)                                                      // Condition for
possible right move
        {
            temp.setBoard(board);
            temp.swap(blankX, blankY, blankX, blankY+1);
            int fn =  (temp.getHn(goal)+gn);
            System.out.println("\nFor Fn = "+fn+" : ");
            temp.display();
            if(fn < minFn)
            {
                minFn = fn;
                next.setBoard(temp.board);
                next.setBlankX(blankX);
                next.setBlankY(blankY+1);
            }

        }
        if(blankX>0)                                                      // Condition for
possible up move
        {
            temp.setBoard(board);
            temp.swap(blankX, blankY, blankX-1, blankY);
            int fn =  (temp.getHn(goal)+gn);
            System.out.println("\nFor Fn = "+fn+" : ");
            temp.display();
            if(fn < minFn)
            {
                minFn = fn;
                next.setBoard(temp.board);
                next.setBlankX(blankX-1);
                next.setBlankY(blankY);
            }

        }
        if(blankX<2)                                                      // Condition for
possible down move
        {
            temp.setBoard(board);
            temp.swap(blankX, blankY, blankX+1, blankY);
            int fn =  (temp.getHn(goal)+gn);
            System.out.println("\nFor Fn = "+fn+" : ");            temp.display();
            if(fn < minFn)
            {
                minFn = fn;
                next.setBoard(temp.board);
                next.setBlankX(blankX+1);                next.setBlankY(blankY);
            }
                     }
        return next;                                                      // return board
with min fn
        }


    public void swap(int i1, int j1, int i2, int j2)                      // Swap tile
values
    {
        String temp = board[i1][j1];        board[i1][j1] = board[i2][j2];
        board[i2][j2] = temp;

    }

    public boolean equals(Board b)                                        // check for
board equality
    {
        for(int i=0; i<3; i++)
        {
            for(int j=0; j<3; j++)
            {
                if(!this.board[i][j].equals(b.board[i][j]))
                {
                    return false;
                }
            }

        }
        return true;

    }

    public int getHn(Board goal)                                            // get hn by
Hamming method
    {
        int hn = 0;
        for(int i=0; i<3; i++)
        {
            for(int j=0; j<3; j++)
            {
                if(!this.board[i][j].equals(goal.board[i][j]))
                {                           hn++;
                }
            }

        }
        return hn;
    } }
```

Output:



```
<terminated> EightPuzzle [Java Application] C:\Program Files\Java\jdk-12.0.2\bin\javaw.e

The given start board is :
        1       2       3
        4       5       6
        7       8       9


 Enter goal Board :

Enter one tile as '-' ie. Blank tile



The given goal board is :
        1       2       3
        4       5       6
        7       8       9


The board is solved as :



Board after 0 moves :
        1       2       3
        4       5       6
        7       8       9

Goal state achieved.
```

```
package Air;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Vector;


public class EightPuzzle {

    private int gn=0;
    private Board start;
    private Board goal;

    public void initStart()
    {
        System.out.println("\n\n Enter start Board : ");
    start=new Board();
    start.initBoard();
        System.out.println("\n\nThe given start board is :
        start.display();
    }

    public void initGoal()
    {
        System.out.println("\n\n Enter goal Board : ");
        goal=new Board();
        goal.initBoard();
        System.out.println("\n\nThe given goal board is :
        goal.display();
    }
```

0 errors, 235 warnings, 0 others (Filter matched 100 of 235 items)

CONCLUSION: 8-puzzle is a simple game consisting of a 3*3 grid containing 9 squares. One of the square is empty. From the given states a program is executed to reach the goal state. It is analyzed and implemented.

FAQS:

1. **How to do you represent knowledge in an uncertain domain.**

Ans: knowledge representation which is best visualized in terms of graphs. It stores the latest truth value of any predicate. The system is developed with the idea that truthfulness of a predicate or agent can change with time, as new knowledge is added or exiting knowledge is updated.

2. **Is 8-puzzle a multi agent or single agent problem? Justify.**

Ans: The eight puzzle, a single agent problem, consists of 3-by-3 square frame which holds eight movable square tiles which are numbered from 1 to 8 .One square is empty, permitting tiles to be shifted. The objective of the puzzle is to find the sequence of tile movements that leads from a starting configuration to a goal configuration

ASSIGNMENT NO: 3

AIM: Implementation of Medical Expert System

PROBLEM STATEMENT Implement any one of the following Expert System , Medical Diagnosis of 10 diseases based on adequate symptoms

PREREQUISITES: Knowledge of Prolog

COURSE OBJECTIVE:

 To understand the concept of Artificial Intelligence (AI)
To learn various peculiar search strategies for AI

COURSE OUTCOMES: Students will be able to,
Design and implement an Medical Expert System

ALGORITHM:

1. First step for execution is install SWI prolog on your machine.
2. Open the prolog editor.
3. Run using go. Command.

- go->.

- Does the patient has the symptom headache? : y.
- Does the patient has the symptom runny nose? : |: n.
- Does the patient has the symptom sore throat? : |: n.
- Does the patient has the symptom abdominal pain? : |: y.
- Does the patient has the symptom poor appetite? : |: y.
- Does the patient has the symptom fever? : |: y.
- Advices and Sugestions:

1: Chloramphenicol
2: Amoxicillin
3: Ciprofloxacin
4: Azithromycin
Please do complete bed rest and take soft diet
because It is suggested that the patient has typhoid true.

CODE:
Output:

CONCLUSION: In this assignment we have successfully implemented. Expert system based on Medical Diagnosis of 10 diseases based on adequate symptoms.

FAQS:

1. What is expert System?

Ans: an Expert system is a computer system that emulates the decision-making ability of a human expert. Expert systems are designed to solve complex problems by reasoning through bodies of knowledge, represented mainly as if–then rules rather than through conventional procedural code.

2. List the components of Expert System.

Ans: An expert system generally consists of four components: a knowledge base, the search or inference system, a knowledge acquisition system, and the user interface or communication system.

3. What is interface engine?

Ans: Inference engine is a component of the system that applies logical rules to the knowledge base to deduce new information.

ASSIGNMENT NO: 4

PROBLEM STATEMENT: Constraint Satisfaction Problem
Implement n-queen problem by using Branch and Bound and Backtracking

Course Objectives:

- To understand the concept of Artificial Intelligence (AI)
- To develop a mind to solve game problems unconventionally with optimality

COURSE OUTCOMES:

Students will be able **to,**
Implement n queen using Backtracking and Branch and Bound Method

ALGORITHM:

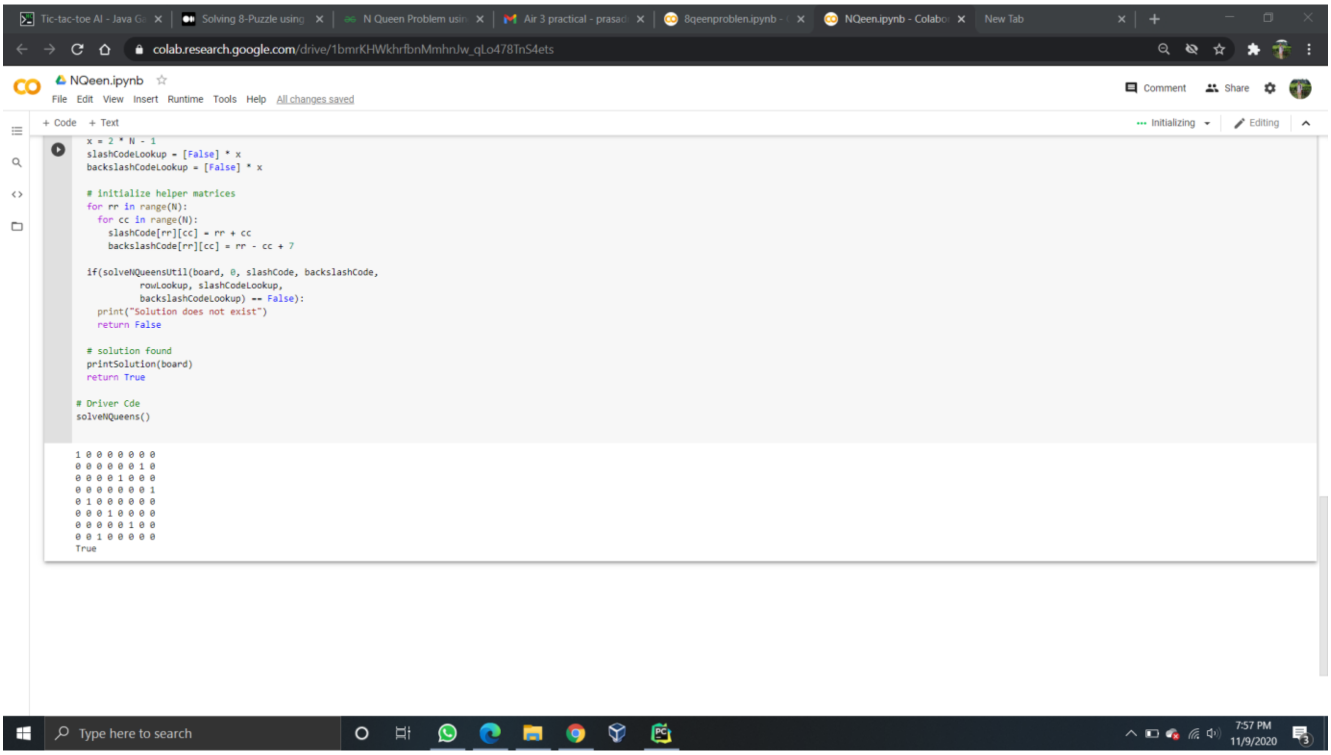
```
1. Begin
2. if all columns are filled, then return true
3. for each row of the board, do
      a. if isValid(board, i, col), then
            i.   set queen at place (i, col) in the board if
                 solveNQueen(board, col+1) = true, then
            ii.  return true
            iii. otherwise remove queen from place (i, col) from board.
4. done
5. return false End
```

CODE:

```
#prasad_code
N = 8

""" A utility function to prsolution """
def printSolution(board):
  for i in range(N):
    for j in range(N):
      print(board[i][j], end = " ")
    print()

""" A Optimized function to check if
a queen can be placed on board[row][col] """
def isSafe(row, col, slashCode, backslashCode,
    rowLookup, slashCodeLookup,
          backslashCodeLookup):
  if (slashCodeLookup[slashCode[row][col]] or
    backslashCodeLookup[backslashCode[row][col]] or
    rowLookup[row]):
    return False
  return True
```

```
def solveNQueensUtil(board, col, slashCode, backslashCode,
          rowLookup, slashCodeLookup,
          backslashCodeLookup):


  if(col >= N):
    return True
  for i in range(N):
    if(isSafe(i, col, slashCode, backslashCode,
        rowLookup, slashCodeLookup,
        backslashCodeLookup)):
      board[i][col] = 1
      rowLookup[i] = True
      slashCodeLookup[slashCode[i][col]] = True
      backslashCodeLookup[backslashCode[i][col]] = True

      if(solveNQueensUtil(board, col + 1,
                slashCode, backslashCode,
                rowLookup, slashCodeLookup,
                backslashCodeLookup)):
        return True

      board[i][col] = 0
      rowLookup[i] = False
      slashCodeLookup[slashCode[i][col]] = False
      backslashCodeLookup[backslashCode[i][col]] = False
  return False
def solveNQueens():
  board = [[0 for i in range(N)]
        for j in range(N)]

  # helper matrices
  slashCode = [[0 for i in range(N)]
          for j in range(N)]
  backslashCode = [[0 for i in range(N)]
            for j in range(N)]

  # arrays to tell us which rows are occupied
  rowLookup = [False] * N

  # keep two arrays to tell us
  # which diagonals are occupied
  x = 2 * N - 1
  slashCodeLookup = [False] * x
  backslashCodeLookup = [False] * x

  # initialize helper matrices
  for rr in range(N):
    for cc in range(N):
      slashCode[rr][cc] = rr + cc
      backslashCode[rr][cc] = rr - cc + 7

  if(solveNQueensUtil(board, 0, slashCode, backslashCode,
            rowLookup, slashCodeLookup,
            backslashCodeLookup) == False):
    print("Solution does not exist")
    return False

  # solution found
  printSolution(board)
  return True

# Driver Cde
solveNQueens()
```

Output:



```
  x = 2 * N - 1
  slashCodeLookup = [False] * x
  backslashCodeLookup = [False] * x

  # initialize helper matrices
  for rr in range(N):
    for cc in range(N):
      slashCode[rr][cc] = rr + cc
      backslashCode[rr][cc] = rr - cc + 7

  if(solveNQueensUtil(board, 0, slashCode, backslashCode,
            rowLookup, slashCodeLookup,
            backslashCodeLookup) == False):
    print("Solution does not exist")
    return False

  # solution found
  printSolution(board)
  return True

# Driver Cde
solveNQueens()
```

```
1 0 0 0 0 0 0 0
0 0 0 0 0 0 1 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 1
0 1 0 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 0
True
```

CONCLUSION: In this assignment we have successfully implemented the n queens using constraint satisfaction problem.

FAQS:

1. Write 4 queens' problem in terms of CST.

Ans: For 4-Queens there are 256 different configurations. We can represent the N-queens as a constraint satisfaction problem. • A Constraint Satisfaction Problem consists of 3 components A set of variables, set of values for each of the variables, set of constraints between various collections of variables. We must find a value for each of the variables that satisfy all of the constraints.

2. Solve 4 queens by using forward checking.

Ans: Forward checking is an extension of backtracking search that employs a “modest” amount of propagation (lookahead). When a variable is instantiated we check all constraints that have only one uninstantiated variable remaining. For that uninstantiated variable, we check all of its values, pruning those values that violate the constraint. Like this, we can solve 4 queens’ problem.

3. Solve Two + Two crypt arithmetic puzzle

Ans: T W O
+ T W O
____
F O U R

There are many solutions for this problem:
938+938=1876
928+928=1856
867+867=1734
846+846=1692
836+836=1672
765+765=1530 734+734=1468

ASSIGNMENT NO: 5

AIM: Implementation of Alpha-Beta pruning

PROBLEM STATEMENT: Implement alpha-beta pruning graphically with proper example and justify the pruning

COURSE OBJECTIVES:

- To understand the concept of Artificial Intelligence (AI)
- To learn various peculiar search strategies for AI
- To develop a mind to solve real world problems unconventionally with optimality

COURSE OUTCOMES:

Students will be able to,

- Implement alpha beta pruning
- Develop a strategy to solve game problems efficiently

ALGORITHM:

```
1. function alphabeta(node, depth, α, β, maximizingPlayer) is if depth = 0 or node is a
   terminal node then
    a. return the heuristic value of node if maximizingPlayer then
    b. value := −∞
    c. for each child of node do
2. value := max(value, alphabeta(child, depth + 1, α, β,
   FALSE)) α := max(α, value)
```

```
        a. if α ≥ β then
                i. break (* β cutoff
                   *) return value
    3. else
        a. value := +∞
        b. for each child of node do
    4. value := min(value, alphabeta(child, depth + 1, α, β,
       TRUE)) β := min(β, value)
        a. if β ≤ α then
                i. break (* α cutoff
                   *) return value
    5. (* Initial call *)
    6. alphabeta(origin, depth, −∞ +∞, TRUE)
```

CODE:

Class for Alphabeta

```
package Air_Prasad;

public class AlphaBeta {
        int scores[]={10,4,5,3,2,8,-1,-9};
        int INFINITY=100;
int MINUSINFINITY=-100;
GUI gui;
        public AlphaBeta()
        {
                gui=new GUI();
        }
        public int minMax(int depth, int nodeNo, boolean isMax, int alpha, int beta)
        {
                if(depth==3)
                {
                        gui.setColor(nodeNo);
                        return scores[nodeNo];

                }

                if(isMax) //maximizing player
                {
                        int bestValue=MINUSINFINITY;
                        for(int i=0;i<2;i++)
                        {
                                int value=minMax(depth+1, nodeNo*2+i, false, alpha, beta);
                                bestValue=Math.max(value, bestValue);
                                alpha=Math.max(bestValue, alpha);
        if(beta<=alpha)
                                        break;
                        }
```

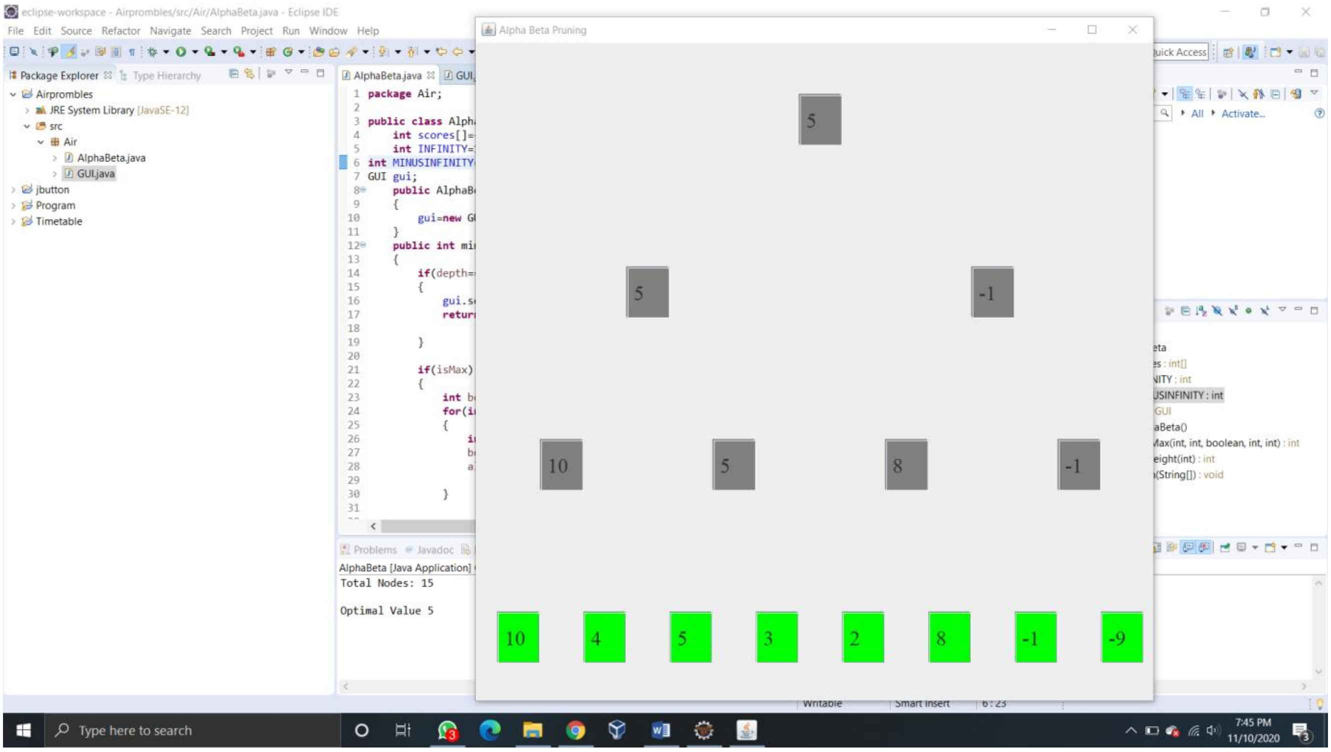
```
            gui.setText(bestValue, depth, nodeNo);
            return bestValue;
        }
        else //minimizing player
        {
            int bestValue=INFINITY;
            for(int i=0;i<2;i++)
            {
                int value=minMax(depth+1, 2*nodeNo+i, true, alpha, beta);
                bestValue=Math.min(bestValue, value);
                beta=Math.min(beta, bestValue);
    if(beta<=alpha)
                    break;
            }
            gui.setText(bestValue, depth, nodeNo);

            return bestValue;
        }
    }

public int getheight(int n)
{
    if(n==1)            return 0;     else
        return 1+getheight(n/2);
}
    public static void main(String[] args) {

        AlphaBeta alphaBeta=new AlphaBeta();

 int height=alphaBeta.getheight(alphaBeta.scores.length);
 alphaBeta.gui.addElements(alphaBeta.scores, height);
        System.out.println();
        System.out.println("Optimal Value "+alphaBeta.minMax(0, 0, true,
alphaBeta.MINUSINFINITY, alphaBeta.INFINITY));
    }

}
```

Class for GUI

```
package Air_Prasad;
import java.awt.*;

import javax.swing.JFrame; import
javax.swing.JTextField;

public class GUI { JFrame frame; int height;
JTextField array[];

public GUI()
{
 frame=new JFrame("Alpha Beta Pruning");
frame.setBounds(0,0,800,800);
frame.setVisible(true);
 frame.setBackground(Color.WHITE);
frame.setDefaultCloseOperation(JFrame.EXI
T_ON_CLOSE);
 frame.setLayout(null);
 }

 public void addElements(int score[], int
height)
 {
          this.height=height;
          int totalNodes=(int) (Math.pow(2,
height+1)-1); //for graph e.g. 15
          System.out.println("Total Nodes:
"+totalNodes);

          array=new JTextField[totalNodes];
          int dataArray[]=new int[totalNodes];

          int start = score.length-1;
//Fill Array with values from score and dummy
values
          int j=0;
          while(start<totalNodes)
          {

         dataArray[start++]=score[j++];
          }
          start=0;
          while(start<score.length-1)
          {
                    dataArray[start++]=-99;
          }

          int rows_of_frame=600/height; //for
finding height of each row start //GUI  things
//200
          int boxheight=60,boxWidth=50;
          Font f=new Font("Serif", Font.PLAIN,
24);
          int treelevel=height;

while(treelevel>=0) //Now actually draw GUI
          {
                    int noOfElements=(int)
Math.pow(2, treelevel);
```

```
                    int
columns_of_frame=800/noOfElements; //100
value



          int offset=columns_of_frame-boxWidth;
//100-50=50
                    offset=offset/2; //25

                    int x=offset;
          int
y=rows_of_frame*treelevel+boxheight;

           start=noOfElements-1; //start index for
data at level

           for(int
i=0,k=start;i<noOfElements;i++,k++)
           {
                    array[k]=new JTextField("
"+dataArray[k]);

array[k].setBounds(x,y,boxWidth,boxheight);


          array[k].setFont(f);

array[k].setBackground(Color.RED);
                    frame.add(array[k]);
                              frame.repaint();

                              x=x+boxWidth+2*offset;
//Important
                    }
                    treelevel--; //reduce level for
drawing
           }

 }

 //for setting
 public void setColor(int node) {
 int offset=(int) (Math.pow(2,height)-1);

array[offset+node].setBackground(Color.GRE
EN);
 }
 public void setText(int ans,int depth,int
nodeNo) {
 int offset=(int) Math.pow(2, depth)-1;
 array[offset+nodeNo].setText(" "+ans);

array[offset+nodeNo].setBackground(Color.G
RAY);
 }

 void setRoot(int ans)
 {
     array[0].setText("  "+ans);
     array[0].setBackground(new Color(150,
150, 150));

 }
}
```

OUTPUT:



```
package Air;

public class Alph
    int scores[]=
    int INFINITY=
int MINUSINFINITY
GUI gui;
    public AlphaB
    {
        gui=new G
    }
    public int mi
    {
        if(depth=
        {
            gui.s
            retur

        }

        if(isMax)
        {
            int b
            for(i
            {
                i
                b
                a

            }
```

```
Total Nodes: 15

Optimal Value 5
```

CONCLUSION: In this assignment we have successfully implemented the alpha beta pruning algorithm

FAQS:

1. What is goal Stack Planning?

Ans: In *goal stack planning*, the problem solver makes use of a *goal stack GS* that contains both subgoals and actions that have been proposed to satisfy those subgoals. It also relies on a database *DB* that describes the *current situation,* and a set of actions described by precondition, add and delete lists.

2. How to represents facts, rules, goals, queries in FOL?

Ans: a. A fact is a predicate expression that makes a declarative statement about the problem domain- likes(john, susie). /* John likes Susie */

b. A rule is a predicate expression that uses logical implication (:-) to describe a relationship among facts.- left_hand_side :- right_hand_side .

c. A goal is a statement starting with a predicate and probably followed by its arguments. Ex, ?− parent(susan, mary). yes

d. A query is a statement starting with a predicate and followed by its arguments, some of which are variables.

Ex, ?- yeye(X, susan). no

3. What are the components of planning?

Ans: States, Goals, Actions, Preconditions, Effect, Finding a Solution, Dead state are the components os a planning system.