**DEPARTMENT OF INFORMATION TECHNOLOGY**

**COURSE CODE: DJ19ITL504 DATE: 22/9/2021**

**COURSE NAME: Artificial Intelligence Laboratory CLASS:TY-IT**

**EXPERIMENT NO.01**

**CO/LO:** Formulate the problem as a state space and select appropriate technique from blind, heuristic or adversarial search to generate the solution.

**AIM / OBJECTIVE:**

Tutorial exercise for

a. Design of Intelligent System using PEAS.

b. Problem Definition with State Space Representation.

**DESCRIPTION OF EXPERIMENT:**

* Student shall design the following agents using PEAS(describe them) and Justify the environments for the same
  + Robot soccer player
  + Shopping for used AI books on the Internet
  + Agent playing Crossword puzzle
  + Bidding on an item in auction.
  + Face recognition for attendance system
  + Google maps
* Students shall formulate state space for the following problem.
  + 5 puzzle problem

**Explanation/Solutions(Design):**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Performance** | **Environment** | **Actuators** | **Sensors** |
| **Robot soccer player** | Playing, Scoring goals, defending from opponents, winning | Playground space, teammates, opponents | Robotic legs/limbs movement | Camera, motion sensors |
| **Shopping for used AI books on the Internet** | Buying books of correct price range, category, authors | Vendor websites, books on the website | Placing an order, filling details | Web scraping |
| **Agent playing Crossword puzzle** | Completing the puzzle correctly, completing the puzzle in less amount of time | Puzzle board | Screen | Keyboard |
| **Bidding on an item in auction.** | Placing bids at the right time on correct objects, knowing when to stop bidding | Auction website, other bidders, Auctioneer, the items | Screen | Getting live updates from websites by scraping |
| **Face recognition for attendance system** | high degree of accuracy in detecting face | Students, teachers and staff | marking the student present. | camera |
| **Google maps** | finding routes correctly between two points, analyzing the traffic and other factors of roads to find best route | satellite images, stored data regarding roads, user | Screen | Keyboard, mic,  touchscreen ,  UI |

**5 PUZZLE:**

3

5

4

1

2

Final configuration :

3

5

4

1

2

Initial configuration :

State space tree**:**

1

3

2

5

4

3

5

4

1

2

3

5

4

1

2

N(incorrect tiles)=4

N(incorrect tiles)=2

3

5

4

1

2

N(incorrect tiles)=1

3

5

4

1

2

3

5

2

4

1

N(incorrect tiles)=0

GOAL STATE

N(incorrect tiles)=2

**Source code:**

#include <stdio.h>

#include <stdlib.h>

int level = 0;

int upmatrix[m][n]={{0, 0, 0}, {0, 0, 0}};

int downmatrix[m][n]={{0, 0, 0}, {0, 0, 0}};

int leftmatrix[m][n]={{0, 0, 0}, {0, 0, 0}};

int rightmatrix[m][n]={{0, 0, 0}, {0, 0, 0}};

int preprematrix[m][n];

// by default the matrix is going to call by reference only

void matrixprint(int matrix[m][n])

{

    // printf("\t\t");

    for (int i = 0; i < 2; i++)

    {

        for (int j = 0; j < 3; j++)

        {

            // matrix[i][j]=4;

            printf("%d ", matrix[i][j]);

        }

        printf("\n");

    }

}

int goalstate(int matrix[m][n], int goal[m][n])

{

    for (int i = 0; i < 2; i++)

    {

        for (int j = 0; j < 3; j++)

        {

            if (goal[i][j] != matrix[i][j])

            {

                return -1;

            }

        }

    }

    return 1;

}

int prevstate(int matrix[m][n], int pre[m][n])

{

    for (int i = 0; i < 2; i++)

    {

        for (int j = 0; j < 3; j++)

        {

            if (pre[i][j] != matrix[i][j])

            {

                return -1;

            }

        }

    }

    return 1;

}

void checkblank(int matrix[m][n], int arr[])

{

    for (int i = 0; i < 2; i++)

    {

        for (int j = 0; j < 3; j++)

        {

            if (matrix[i][j] == -1)

            {

                arr[0] = i;

                arr[1] = j;

            }

        }

        printf("\n");

    }

}

void moveup(int matrix[m][n])

{

    int temp;

    int pos[2];

    checkblank(matrix, pos);

    temp = matrix[pos[0] - 1][pos[1]];

    matrix[pos[0] - 1][pos[1]] = -1;

    matrix[pos[0]][pos[1]] = temp;

}

void movedown(int matrix[m][n])

{

    int temp;

    int pos[2];

    checkblank(matrix, pos);

    temp = matrix[pos[0]+1][pos[1]];

    matrix[pos[0]+1][pos[1]] = -1;

    matrix[pos[0]][pos[1]] = temp;

}

void moveleft(int matrix[m][n])

{

    int temp;

    int pos[2];

    checkblank(matrix, pos);

    temp = matrix[pos[0]][pos[1]-1];

    matrix[pos[0]][pos[1]-1] = -1;

    matrix[pos[0]][pos[1]] = temp;

}

void moveright(int matrix[m][n])

{

    int temp;

    int pos[2];

    checkblank(matrix, pos);

    temp = matrix[pos[0]][pos[1]+1];

    matrix[pos[0]][pos[1]+1] = -1;

    matrix[pos[0]][pos[1]] = temp;

}

void copy( int newcopy[m][n],int matrix[m][n])

{

    for (int i = 0; i < 2; i++)

    {

        for (int j = 0; j < 3; j++)

        {

            newcopy[i][j] = matrix[i][j];

        }

    }

}

int solvematrix(int matrix[m][n], int goal[m][n])

{

    int u=0,l=0,r=0,d=0;

    level = level + 1;

    int prev[m][n];

    copy(prev,matrix);

    int pos[2];

    int result;

    checkblank(matrix, pos);

    printf("level %d", level);

    if (pos[0] == 1)

    {

        moveup(matrix);

        if(prevstate(matrix,preprematrix)!=1)

        {

            u=1;

        }

        copy(upmatrix,matrix);

        matrixprint(matrix);

    }

    if (goalstate(matrix, goal) == 1)

    {

        printf("\n End Of The Tree\n");

        return 1;

    }

    copy(matrix,prev);

    if(pos[0]==0)

    {

        if(prevstate(matrix,preprematrix)!=1)

        {

               d=1;

        }

        movedown(matrix);

        copy(downmatrix,matrix);

        matrixprint(matrix);

    }

    if (goalstate(matrix, goal) == 1)

    {

        printf("\n End Of The Tree\n");

        return 1;

    }

    copy(matrix,prev);

    if(pos[1]==0 || pos[1]==1 )

    {

        if(prevstate(matrix,preprematrix)!=1)

        {

               r=1;

        }

        moveright(matrix);

        copy(rightmatrix,matrix);

        matrixprint(matrix);

    }

    if (goalstate(matrix, goal) == 1)

    {

        printf("\n End Of The Tree\n");

        return 1;

    }

    copy(matrix,prev);

    if(pos[1]==1 || pos[1]==2)

    {

        if(prevstate(matrix,preprematrix)!=1)

        {

               l=1;

        }

         moveleft(matrix);

         copy(leftmatrix,matrix);

         matrixprint(matrix);

    }

    if (goalstate(matrix, goal) == 1)

    {

        printf("\n End Of The Tree\n");

        return 1;

    }

    copy(preprematrix,prev);

     if(u!=0)

    {

        solvematrix(upmatrix,goal);

    }

     if(d!=0)

    {

        solvematrix(downmatrix,goal);

    }

     if(l!=0)

    {

        solvematrix(leftmatrix,goal);

    }

     if(r!=0)

    {

        solvematrix(rightmatrix,goal);

    }

    return 0;

}

int main()

{

    int matrix[2][3] = {{-1, 2, 3}, {1, 4, 5}};

    int goal[2][3] = {{1, 2, 3}, {4, 5, -1}};

    int temp[2];

    solvematrix(matrix, goal);

    checkblank(matrix, temp);

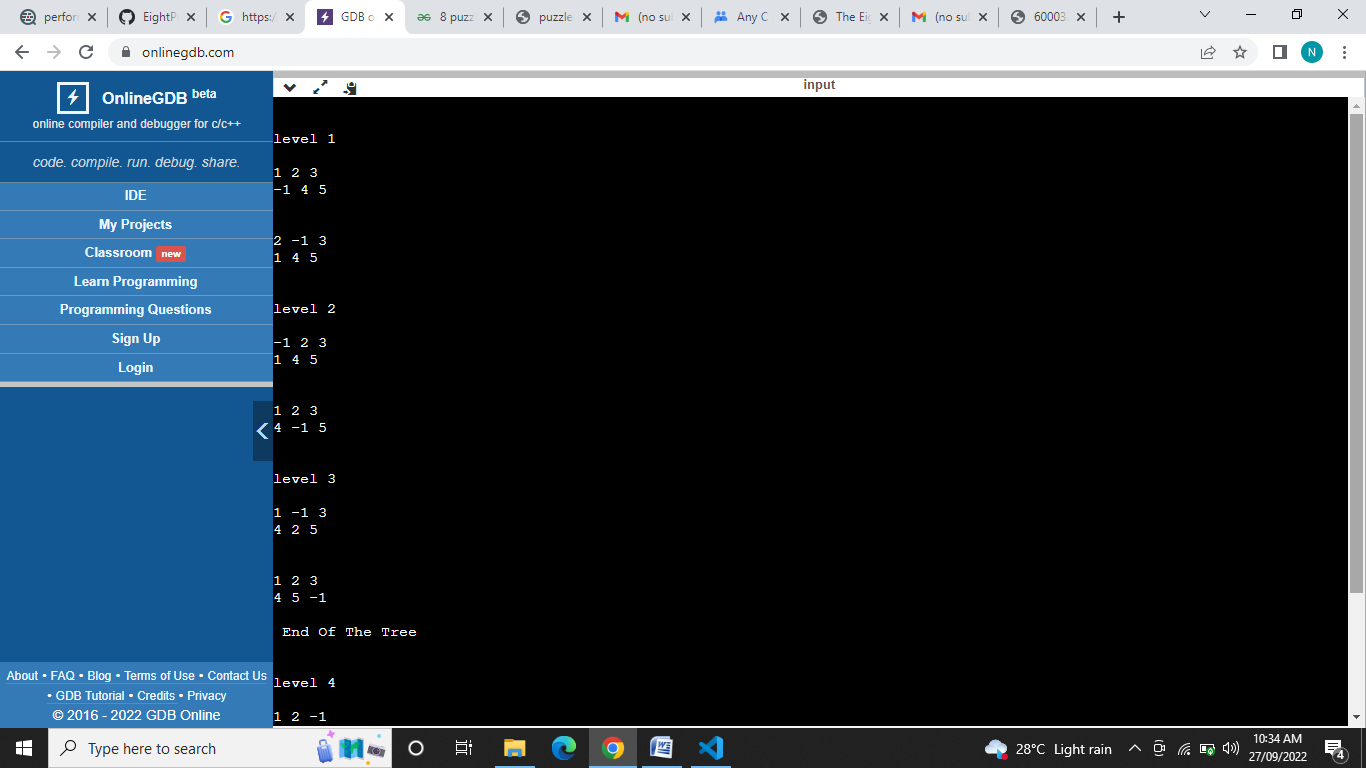
    // matrixprint(matrix);

    // matrixprint(goal);

    return 0;

}

Output:



**CONCLUSION:**

Thus the task environment of agents can be described for their performance measure, environment , actuators ,sensors using PEAS and 5 puzzle problem can be represented in its state space tree form using a bounding function/ heuristic.