classmate
Date
Page

1: TTT using Magic Square:

considers the given pre-sets:

magic square: $\begin{bmatrix} 8 & 3 & 4 \\ 1 & 5 & 9 \\ 6 & 7 & 2 \end{bmatrix}$

move: move to be played
mc1: move count for comp
mc2: move count for player
c : check for win
comp[] : Array for computer moves
play[] : Array for player moves
arr[9] : Board

STEP 1: Choose whether player starts or comp starts.

STEP 2: If player - move:
player may choose any one of the available move.

If comp - move:
If (mc1 == 1):
if (center is available):
comp plays center.
else
comp plays any of the corner moves

else if (mc1 == 2):
comp plays any of the corner moves

else
if (win - move == 1):
comp plays win move

```
        else if ( block_move == 1)
            comp plays block move

        else
            comp plays random move
```

STEP 3:

```
for ( player):
    if ( win_condition == 1)
        print ( Player wins)
        Goto STEP 6

for (comp):
    if ( win_condition == 1)
        print (computer wins)
        GOTO STEP 6
```

STEP 4: Repeat steps 2,3 while (board != Full)

STEP 5:

```
if ( board == FULL)
    print ( Game Draw)
```

STEP 6: END

```
win_move:
    a = 15 - comp [i-1] - comp [i-2]
    if (a<9)
    return a

block_move:
    a = 15 - play [i-1] - play [i-2]
    if (a<9)
    return a
```

```
win_condition:
    i=0
    while (i<9)
        if (arr[i] && arr[i+1] && arr[i+2])
            return 1
        i = i+3
    i=0
    while (i<3)
        if (arr[i] && arr[i+3] && arr[i+6])
            return 1
        i++
    i=0
    if (arr[i] && arr[i+2] && arr[i+4])
        return 1

    if (arr[i] && arr[i+4] && arr[i+8])
        return 1

    return 0
```

4 4

2. TTT using MinMax

Consider the given Pre-sets:

move : move to be played
b[3][3] : board , ass[9] : board
Max : Comp , Min : Player

STEP 1: Choose whether player starts or computer starts

STEP 2: if player_move:
player may choose any 1 of the valid moves.

if comp_move:
```
best_val = -1000
for (i=0; i<5; i++)
    for (j=0; j<5; j++)
        if (b[i][j] == ' ')
            b[i][j] = move
            mov_val = minmax (board, isMax)
            if (mov_val > best_val)
                b[i][j] = ' '
                comp plays mov_val
                best_val = mov_val
```

STEP 3: for (players):
```
if (win_condition == 2)
    print (Player wins)
    GOTO STEP 6
```

classmate
Date
Page

```
for (comp):
    if (win - condition == 1)
        print (computer wins)
        GOTO STEP6
```

STEP 4: Repeat steps 2,3 while (board != FULL)

STEP 5: if (board == FULL)
        print (Game Draw)

STEP 6: END

```
minmax:
    score = heuristic (board)
    if (score == 10)
        return score
    if (score == -10)
        return score
    if (is max)
        best = -1000
        for (i=0; i<5; i++)
            for (j=0; j<5; j++)
                if (b[i][j] == '')
                    b[i][j] = move
                    best = max (best, minmax (board, !isMax))
                    b[i][j] = ''
        return best
    else
        best = 1000
        for (i=0; i<5; i++)
            for (j=0; j<5; j++)
```

```
            if (b[i][j] == '')
                b[i][j] = move
                best = min (best, minmax (board, isMax))
                b[i][j] = ''
        return best.

heuristic:
    i = 0
    while (i<9)
        if (arr[i] && arr[i+1] && arr[i+2])
            return 10
        else
            return -10
        i = i+3
    i=0
    while (i<3)
        if (arr[i] && arr[i+3] && arr[i+6])
            return 10
        else
            return -10
        i++
    i=0
    if (arr[i] && arr[i+2] && arr[i+4])
        return 10
    else
        return -10

    if (arr[i] && arr[i+4] && arr[i+8])
        return 10
    else
        return -10
```

classmate
Date
Page

```
win_condition:
    i=0
    while (i<9)
        if (arr[i] && arr[i+1] && arr[i+2])
            return 1
        i = i+3
    i=0
    while (i<3)
        if (arr[i] && arr[i+3] && arr[i+6])
            return 1
        i++
    i=0
    if (arr[i] && arr[i+2] && arr[i+4])
        return 1
    if (arr[i] && arr[i+4] && arr[i+8])
        return 1

return 0
```

CLASSMATE
Date
Page

3. Create Magic Square

Consider the following presets:

arr [i] [j] = board
n : user value for size of matrix
Algorithm only for odd values of 'n'
val : value to be added next

STEP 1: Accept value of 'n' from user
Create a 2D matrix with n rows and columns
Set entire matrix to '0'.

STEP 2: First value val = 1 is set on arr [i] [j] where:
i = 0
j = (n+1)/2

STEP 3:
```
temp 1 = i
temp 2 = j
i--
if (i == -1)
    i = n - 1
j++
if (j == n)
    j = 0
if (arr [i] [j] == 0)
    arr [i] [j] = val
else
    i = temp 1 + 1
    j = temp 2
    arr [i] [j] = val
```

classmate
Date
Page

STEP 4: Repeat Step 3 while (board != FULL)

STEP 5: Print final board

END

classmate
Date
Page

4. N-Queens

Consider the following Presets

arr [][] : board
q = queen number
n = No of queens to be placed.

STEP 1: Accept value of 'n' from user.
create a 2D matrix with 'n' rows and columns
set entire matrix to '0'.

STEP 2:
```
i = 0 , j = q - 1
if (arr [i] [j] = = 0)
    arr [i] [j] = q
    set all board tiles in the row, column and
    diagonal of the queen (q) to -q.
    q++
else
    i++
if (i==n && j = q-1)
    set all board tiles in the row, column and
    diagonal of Queen (q-1) to 0
    Remove queen (q-1) and set tile to -1
    q--
```

STEP 3: Repeat step 2 while (q != n+1)

STEP 4: Display Board

STEP 5: END

classmate
Date
Page

5. Water Jug (BSF)

Consides the following presets
J1 : Max capacity of Jug 1
J2 : Max capacity of Jug 2
s1 : value of Jug 1 in Goal state
s2 : value of Jug 2 in Goal state
open[] : nodes whos children are not calculated
close[] : nodes whos children are calculated
x : current value of Jug 1
y : current value of Jug 2
Initially both jugs are empty.
All possible rules have been declared.

STEP 1: Accept value of J1, J2, S1 and S2 from user
Set Root node with x=0 , y=0

STEP 2: Apply all possible rules on root node to find all possible child nodes
if (child_node == Goal)
    GOTO STEP 5
update open and close list

STEP 3: Find ~~Leaf~~ Left most Leaf Node of Root
Apply all possible rules to find all possible child nodes
if (child_node == Goal)
    GOTO STEP 5
else
    update close and open list
    ~~find~~ backtrack to parent node
    find next left most node

Classmate
Date
Page

STEP 4: Repeat step 3 while Goal State is not reached

STEP 5: Print Tree traversal route from Root to Goal node

END

classmate
Date
Page

6. Water Jug (DSF)

Presets same as 5.

STEP 1: Accept value of J1, J2, S1, S2 from user
Set root node with x = 0, y = 0

STEP 2: Repeat while (child_node == UNIQUE)
Find child_node of parent
if (child_node == Goal)
GOTO STEP 4
else
update open and close list
end loop

STEP 3: Backtrack to parent node of current leaf node.
Find next child_node of parent
GOTO STEP 2

STEP 4: Print Tree Traversal route from root to Goal
END

7. 8-Puzzle (Hill climbing)

consider the following presets:
Intial and goal state are hard coded
objective function: no. of tiles in correct position
obj: objective function value of given node
c = changed node
All possible board moves have been declared.

STEP 1: Set Root node as Initial state.

STEP 2: Repeat while (node(obj) != 8 || local maxima)

Apply all possible board moves to find all child nodes
Calculate obj value of all child nodes.

```
for (i=0; i<child_nodes; i++)
   c=0
   if (child(obj) == 8)
      print (Goal State Achieved)
      GO TO STEP 3
   if (child(obj) > parent(obj))
      parent = child
      c = 1
   if (c==0)
      print (Local Maxima!)
      GO TO STEP 3
End of loop
```

STEP 3: END

classmate
Date
Page

8. 8-Puzzle (Best)

Consider the following pre-sets:
Initial and goal state are pre-set / hard coded
Heuristic function: No. of tiles in correct position.
h: heuristic value of given node
open[]: list of nodes whos children are not found
close[]: list of nodes whos children are found
All possible board moves have been declared.

STEP 1: set Root node as initial state.

STEP 2: Repeat while (node(h) != 8)
- update open and close list
- Apply all possible board moves to find all child nodes of current_node
- Calculate heuristic value of all child nodes

```
for(i=0; i< all_node; i++)
    if (node(h) == 8)
        print (Goal State Achieved)
        Goto STEP 3
    if (node(h) > current(h))
        current = node
```

end of loop

STEP 3: Print tree traversal from Root to Goal

END

9. 8-Puzzle (A*)

consider the following pre-sets:
Initial and Goal state are hard coded
Evaluation function $f(n) = g(n) + h(n)$
$g(n)$: Distance travelled from root
$h(n)$: Heuristic function value misplaced
• Heuristic function: No. of tiles misplaced
open[] : list of nodes whos children are not found
close[] : list of nodes whos children are found.

STEP 1: set Root node as Initial State.

STEP 2: Repeat while (node (h) != 0)
update open and close list
apply all possible board moves to find all child_nodes of current_node
calculate evaluation function value of all child nodes
for (i=0; i < all_nodes; i++)
if (node (h) == 0)
print (Goal State Achieved)
GOTO STEP 3
if (node (f) < current (f))
current = node
end of loop

STEP 3: Print tree traversal from root to goal
END

classmate
Date
Page

10. City Distance (Best)

Consider the given Pre-sets:
Intial and Goal cities are hard coded.
Edge value indicates straight line distance
Heuristic function: shortest distance
h: heuristic value of given node
open[]: list of nodes whos childrens are not found
close[]: list of node whos childen are found.

STEP 1: Create a weighted graph with each city as a vertex and distance between two cities as weight of that edge

STEP 2: Set origin_city as root

STEP 3: Repeat while (node(h) != 0)
Find all cities connected to current_node from graph
Calculate heuristic value of each node
for (i=0; i < all_nodes; i++)
if (node(h) == 0)
print (Destination Reached)
GOTO STEP 4
if (node(h) < current(h)
current = node
update open and close list
end of LOOP

STEP 4: Print tree traversal route from origin to destination and total distance travelled
END

11. City Distance (A*)

Consider the following presets:
Initial and Goal city is hard coded
Edge value indicates straight line distance
Evaluation function : $f(n) = g(n) + h(n)$
$g(n)$: Distance travelled from origin
$h(n)$: Distance from destination
open[] : list of nodes whos childrsen are not found
close[] : list of nodes whos children are found

STEP 1: Create a weighted graph with each city as a vertex and distance between two cities as weight of edge.

STEP 2: Set origin_city as root node

STEP 3: Repeat while (node(h) != 0)
Find all cities connected to current_node from graph
Calculate evaluation function value of each node
for (i=0; i< all_nodes ; i++)
    if (node(h) == 0)
        print (Destination Reached)
        GO TO STEP #4
    if (node(f) < current(f))
        current = node
update open and close list
end of loop

STEP 4: Print tree traversal from origin to destination and total distance travelled
END

classmate
Date
Page

12. Robot (Best)

Consider the given pre-sets:
Initial and goal positions are hardcoded.
Heuristic function: manhattan distance
open[]: list of nodes whos children are not found
close[]: list of nodes whos children are found.
h: heuristic value of a node.

STEP 1: Set initial position as root node.

STEP 2: Repeat while (node (h) != 0)
update open and close list
Find all tiles that can be reached from current_node
Calculate heuristic value of child_nodes.
for (i=0; i< all_nodes; i++)
if (node (h) == 0)
print (Goal position Reached)
GOTO STEP 3
if (node (h) < current (h))
current = node
else
current = open [0]
end of loop.

STEP 3: Print robot navigation path
END

CLASSMATE
Date
Page

13. Robot (A*)

consider the following pre-sets:
Initial and goal position is hard coded
Evaluation function $f(n) = g(n) + h(n)$
$g(n)$: distance travelled from initial tile
$h(n)$: manhattan distance to goal tile
open[] : list of nodes whos children are not found
close[] : list of nodes whos children are found.

STEP 1: Set initial position as root node.

STEP 2: Repeat while (node (h) != 0)
update open and close list
Find all possible tiles that can be reached from current - node.
calculate evaluation function value of all child - nodes.
for (i=0; i<= all_nodes ; i++)
if (node (h) == 0)
print (Goal Reached)
GO TO STEP 3
if (node (f) < current (f))
current = node
else
current = open [0]
end of loop

STEP 3: Print robot navigation path
END

classmate
Date
Page

14. MAP Colouring :

consider the following presets:
col[] : array containing all colours to be used.
open[] : all nodes that are not yet coloured
close[] : all nodes that are coloured

STEP 1: Create a graph with each map element as a vertex and connect neighbouring elements using edges.

STEP 2: Repeat while (open[] != EMPTY)
for (i=0; i<open; i++)
- find all nodes connected to open[i]
- Eliminate all used colours
- Choose colour for 'i' from remaining available colours.

end of loop

STEP 3: if (open == EMPTY)
GOTO STEP 4

else
print("could not colour entire map")
GOTO STEP 5

STEP 4: Print colour sequence of all Map Elements

STEP 5: END