AI Pseudocodes.

MAP coloring.

col[] : array containing all the colors to be used
open[] : all the nodes that are not colored yet
close[] : all the nodes that are colored.

STEP1: Create a graph with each map element as a vertex and connect the neighboring elements using edges.

STEP2: Repeat while (open[] != EMPTY)
for (i=0 ; i < len.open ; i++)
i) - find all nodes connected to open[i]
ii) Eliminate all used colors
iii) choose color for 'i' from the remaining available colors

end of loop

STEP3: if (open == EMPTY)
goto STEP 4:

else
print (could not color entire map)
goto STEP 5:

STEP4: Print color sequence of all map elements

STEP5: END

WATER JUG - BFS.

Preconditions

J1 : Max capacity of jug 1
J2 : Max capacity of jug 2
G1 : Value of Jug 1 in goal state
G2 : Value of Jug 2 in goal state

open[] : nodes whose children are not calculated
close[] : children are calculated
x_1 : current value of Jug 1
x_2 : current value of Jug 2

- Initially both jugs are empty
- rules have been declared

STEP 1: Accept values of J_1, J_2, G_1, G_2 from the user & set root node as $x_1 = 0$ & $x_2 = 0$.

STEP 2: i) Apply all rules on root node to find the possible child nodes

ii) if (child_node == GOAL)
goto STEP 4

iii) update open[], close[].

STEP 3: i) Find left most leaf node of root, repeat till goal state is reached.

ii) Apply all rules to find all possible child nodes

~~City Distance~~ 8 PUZZLE - Hill Climbing.

Pseudo

- initial and goal states are predefined
- misplaced_tiles() - no. of misplaced tiles from the goal state

STEP 1: Set the root node as initial state.

STEP 2: Repeat while (node(misplaced_tile)!=0 || local maxima)

i) get successors by checking neighboring empty tiles
ii) calculate heuristic value for each successor

```
for(i=0; i < child_nodes ; i++)
  { change=0
    if(child(misplaced_tile)==0)
        print(goal state achieved)
        goto STEP 3:

    if (child(misplaced_tile) <parent(misplaced_tile)
        parent = child
        change = 1

    if (change ==0)
        print (Local maxima)
        goto STEP 3:
```

end of loop.

STEP 3: END

Robot Navigation A*

board[]: a(9x5) matrix representing the robots arena
class Node: for (x,y) with parent & manhattan dist.
open[]: open priority queue to store path.
close[]: normal queue to store shut down nodes

STEP1: define initial node with the initial coordinates of the robot and get the manhattan_distance(). as well as the goal state.

STEP2: Start search with initial node

```
search (init) {
    open.append(init)
    while (open not EMPTY)
    {
      if (open.top == GOAL)
        return NODE
      else
i)  get possible successors
ii) add best choice to open if its
    not in close[]
    best choice . g(h) = f(h) + h(n)
}
```

STEP3: Print all parents of the node to get the path

STEP4: END.

iii) if (child_node == GOAL)
goto STEP 4
else
update close[], open[]
Backtrack to parent node
find next left most node.

STEP4: Print open[] to see the path from root to goal.

WATER JUG - DFS

preconditions: same as BFS.

STEP 1: Take J_1, J_2, G_1, G_2 as input and set root node as $x_1 = 0$ & $x_2 = 0$

STEP 2: Repeat while (child_node == UNIQUE)
{
find child_node of parent
if (child_node == GOAL)
goto STEP 4
else
update open[], close[]
end loop.

STEP 3: Backtrack to parent of current leaf node & find next child_node. . goto STEP 2:

STEP 4: Print open[] for path
END.

STEP3: Repeat step 2 till (q != n+1)

STEP4: display board

STEP5: END.

Magic Square

pre conditions:
N = size of matrix
n = possible number to be filled
i,j = cell co-ordinates

STEP1: set i=0, j=N/2 initially (to place the numb
in the top row, comb

arr[i][j] = n, n+1

STEP2: newti = (i-1) % N
newtj = (j+1) % N

if (X(newti, newtj))
i++;
else
i,j = newti, newtj

TEP3: print arr.

N Queens.

pre conditions:

arr[][] : board of '0'
n : no. of queens to be placed
qno. : queen number.

STEP 1: Accept the value of 'n' from user.

Create a 2D matrix with n x n as dimensions and set the values to zero. ('0')

STEP 2: i = 0, j = q - 1

if (arr[i][j] == 0)
  i) arr[i][j] = q
  ii) set all board tiles in the row, column and diagonal of queen (q) to '-q'
  q++

else
  i++
  if (i == n && j = q-1)
    i) set all board tiles in the row, column and diagonal of queen (q-1) to
    ii) Remove queen (q-1) & set tile to -1

STEP3: Repeat step 2 till (q != n+1)

STEP4: display board

STEP5: END.