A* Algorithm

Pseudo Code and Flow Chart

Algorithm 1 A-Star

```
procedure SEARCH
    Initialize lists $open$ $closed$ $action$ $policy$
    $fcost = gcost + hcost$
    Add $[fcost[start], hcost[start], gcost[start], start]$ in $open$
    while not $found$ and not $resign$ do
        if empty $open$ then
            Return $resign$
        end if
        Inverse Sort $open$
        Pop minimum value $tuple$ from $open$
        if $tuple$ is $goal$ then
            Set $found$ to True
        end if
        Explore $neighbours$ of $tuple$ not in $closed$
        Add $neighbours$ to $open$
        Add $neighbours$ to $closed$
        Save $action$ for $neighbours$
    end while
    if thenfound
        Back-propagate from $goal$ until $start$
        Save $policy$ in each back-propagate iteration
        Print $policy$
    end if
end procedure
procedure EUCLIDEAN
    for each element $i$ in $Grid$ do
        $hcost[element] = D * (xDistance[element][goal] + yDistance[element][goal])$
    end for
end procedure
procedure MANHATTAN
    for each element $i$ in $Grid$ do
        $hcost[element] = D * sqrt(xDistance[element][goal] * *2 + yDistance[element][goal] * *2)$
    end for
end procedure
```

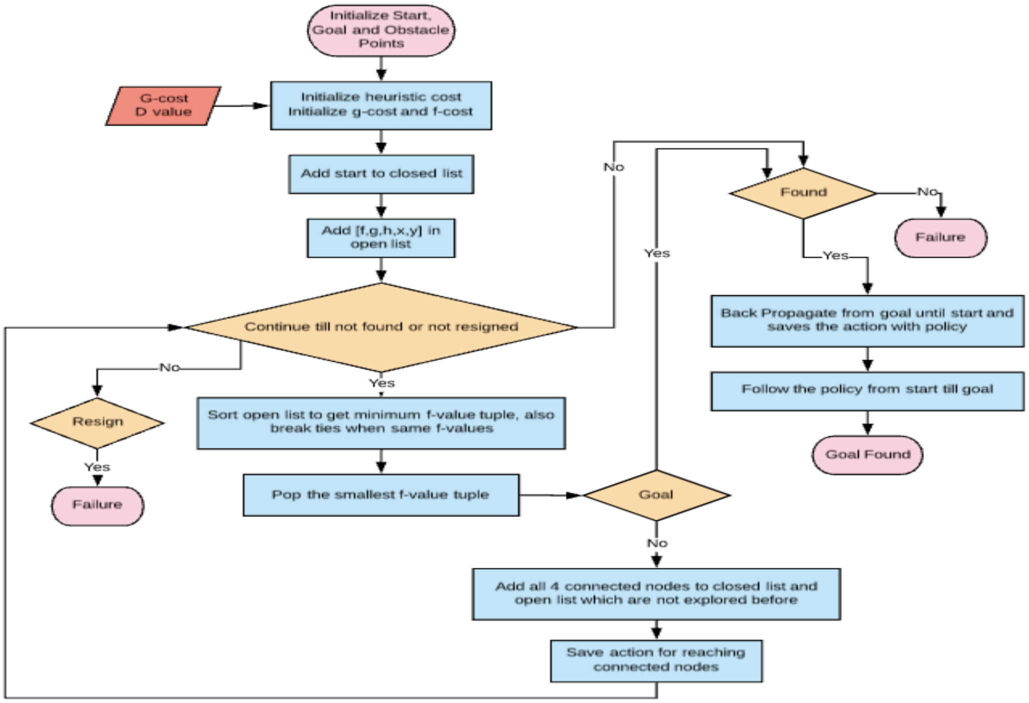
Grid World Functions

#Functions for Grid Creation
# Grid Initialization
createGrid()
loadImages()

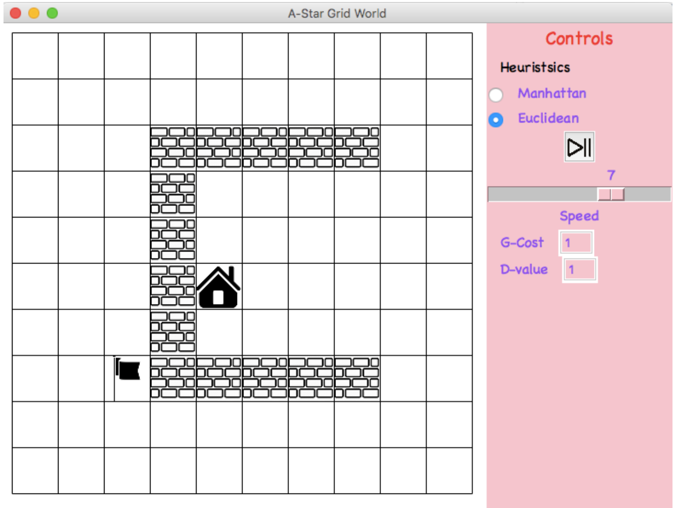
# Draws respective images at location x,y
drawStart(x,y)
drawTarget(x,y)
drawWall(x,y)

# Control Functions
selectHeuristic()
play()
g_cost()
get_D - D used in heuristics
get_sleep() - Speed Bar

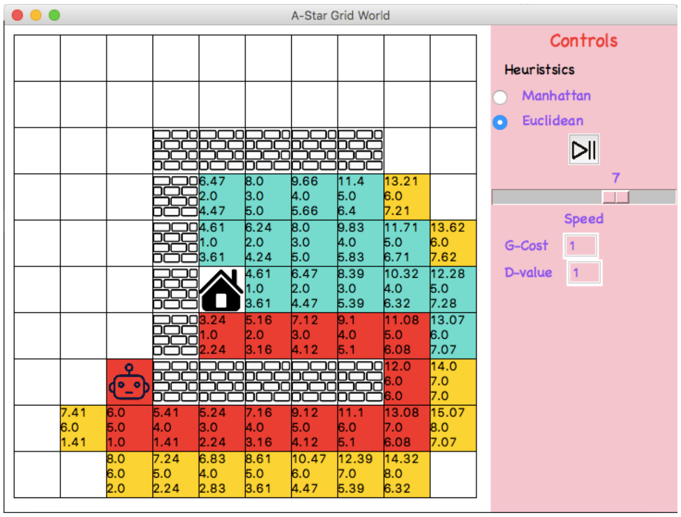
Functions at run time
drawRobot(x,y) - Moves robot at position (x,y)
drawText(x,y,f,g,h,c) - Write cost values at position(x,y)


Initialize Start, Goal and Obstacle Points
G-cost D value
Initialize heuristic cost Initialize g-cost and f-cost
Add start to closed list
Add [f,g,h,x,y] in open list
Continue till not found or not resigned
No
Yes
No
Resign
Yes
Failure
Sort open list to get minimum f-value tuple, also break ties when same f-values
Pop the smallest f-value tuple
Goal
Yes
No
Add all 4 connected nodes to closed list and open list which are not explored before
Save action for reaching connected nodes
Found
No
Failure
Yes
Back Propagate from goal until start and saves the action with policy
Follow the policy from start till goal
Goal Found

A* Example



○ Manhattan
● Euclidean
7
Speed
G-Cost 1
D-value 1



○ Manhattan
● Euclidean
7
Speed
G-Cost 1
D-value 1

				6.47 2.0 4.47	8.0 3.0 5.0	9.66 4.0 5.66	11.4 5.0 6.4	13.21 6.0 7.21	
				4.61 1.0 3.61	6.24 2.0 4.24	8.0 3.0 5.0	9.83 4.0 5.83	11.71 5.0 6.71	13.62 6.0 7.62
					4.61 1.0 3.61	6.47 2.0 4.47	8.39 3.0 5.39	10.32 4.0 6.32	12.28 5.0 7.28
				3.24 1.0 2.24	5.16 2.0 3.16	7.12 3.0 4.12	9.1 4.0 5.1	11.08 5.0 6.08	13.07 6.0 7.07
								12.0 6.0 6.0	14.0 7.0 7.0
	7.41 6.0 1.41	6.0 5.0 1.0	5.41 4.0 1.41	5.24 3.0 2.24	7.16 4.0 3.16	9.12 5.0 4.12	11.1 6.0 5.1	13.08 7.0 6.08	15.07 8.0 7.07
		8.0 6.0 2.0	7.24 5.0 2.24	6.83 4.0 2.83	8.61 5.0 3.61	10.47 6.0 4.47	12.39 7.0 5.39	14.32 8.0 6.32	

Example Setting:

In above example, Robot starts exploring from the home. Turquoise Area shows the explored area and Gold area shows the unexplored area, but which is in open list. Robot follows the path from home to flag following the red area. Values shown at each cell is in the order f_cost -> g_cost -> h_cost.

```
G-Values
[[  9.    8.    7.    6.    5.    6.    7.    8.    9.   10.]
 [  8.    7.    6.    5.    4.    5.    6.    7.    8.    9.]
 [  7.    6.    5.  inf   inf   inf   inf   inf    7.    8.]
 [  6.    5.    4.  inf    2.    3.    4.    5.    6.    7.]
 [  5.    4.    3.  inf    1.    2.    3.    4.    5.    6.]
 [  4.    3.    2.  inf    0.    1.    2.    3.    4.    5.]
 [  5.    4.    3.  inf    1.    2.    3.    4.    5.    6.]
 [  6.    5.    4.  inf   inf   inf   inf   inf    6.    7.]
 [  7.    6.    5.    4.    3.    4.    5.    6.    7.    8.]
 [  8.    7.    6.    5.    4.    5.    6.    7.    8.    9.]]
```

```
H-Values
[[ 7.28  7.07  7.    7.07  7.28  7.62  8.06  8.6   9.22  9.9 ]
 [ 6.32  6.08  6.    6.08  6.32  6.71  7.21  7.81  8.49  9.22]
 [ 5.39  5.1   5.     inf   inf   inf   inf   inf  7.81  8.6 ]
 [ 4.47  4.12  4.     inf  4.47  5.    5.66  6.4   7.21  8.06]
 [ 3.61  3.16  3.     inf  3.61  4.24  5.    5.83  6.71  7.62]
 [ 2.83  2.24  2.     inf  2.83  3.61  4.47  5.39  6.32  7.28]
 [ 2.24  1.41  1.     inf  2.24  3.16  4.12  5.1   6.08  7.07]
 [ 2.    1.    0.     inf   inf   inf   inf   inf  6.    7.  ]
 [ 2.24  1.41  1.    1.41  2.24  3.16  4.12  5.1   6.08  7.07]
 [ 2.83  2.24  2.    2.24  2.83  3.61  4.47  5.39  6.32  7.28]]
```

```
F-Values
[[ 16.28  15.07  14.    13.07  12.28  13.62  15.06  16.6   18.22  19.9 ]
 [ 14.32  13.08  12.    11.08  10.32  11.71  13.21  14.81  16.49  18.22]
 [ 12.39  11.1   10.      inf    inf    inf    inf    inf  14.81  16.6 ]
 [ 10.47   9.12   8.      inf   6.47   8.     9.66  11.4   13.21  15.06]
 [  8.61   7.16   6.      inf   4.61   6.24   8.     9.83  11.71  13.62]
 [  6.83   5.24   4.      inf   2.83   4.61   6.47   8.39  10.32  12.28]
 [  7.24   5.41   4.      inf   3.24   5.16   7.12   9.1   11.08  13.07]
 [  8.     6.     4.      inf    inf    inf    inf    inf  12.    14.  ]
 [  9.24   7.41   6.     5.41   5.24   7.16   9.12  11.1   13.08  15.07]
 [ 10.83   9.24   8.     7.24   6.83   8.61  10.47  12.39  14.32  16.28]]
```

Note: Heuristic can be Euclidean or Manhattan. For Tie breaking (same f-cost), scale the heuristic values by a factor of D, which will give different f-costs and robot will explore less to find the optimal path. Also if we want to give g-cost more weight over h-cost, then g-value is used to scale the actual cost of going from one node to another.