1 – Initial state of the brick is the representation in a 2D array with 1 cell or 2 cells depending on the current shape of the brick (whether it is horizontal or vertical) with start state. Vertical shape can be represented by one cell whereas horizontal shape can be represented by 2 cells. Step cost is equal to 1 because a brick has 4 possible successors next to it which are up, down, left and right adjacent cells. Successors depend on the brick shape too. If brick is horizontal and it looks right or left, then upper and lower cells and right and left 1 cell will be the successor of the brick. Another case is brick is horizontal but looking up or down and it will have 2 cells corresponding to the right and left successors and 1 cell corresponding to the up and down successors. If brick is vertical it will have 4 successors containing 2 cells in every possible direction (up, down, left, right). For a brick to have a successor corresponding cells must hold 'O' or 'G'. A brick cannot have a successor containing 'X' character. The goal state will be when the coordinate of the brick is on the goal coordinate and it has a vertical shape. If it is horizontal it will keep searching for the goal state till it is vertical.

3 – The heuristic function will be the Manhattan distance function. Each successor will be put in a frontier depending on its distance from the goal. The formula for vertical brick is: abs (goal.x – brick.x) + abs (goal.y – brick.y). For a horizontal one:
Min (abs (goal.x – brick.x[0]) + abs (goal.y – brick.y[0]) , abs(goal.x – brick.x[1]) + abs (goal.y – brick.y[1])). The Manhattan distance function is admissible because it always puts a successor into frontier based on its f(n) = g(n) + h(n) where h finds the distance to the goal. Thus, it finds the optimal solution and never overestimates the number of moves to solve the problem.

5 –

There is a considerable difference between A* and Uniform Cost Search algorithms in terms of memory and time consumption. These results are not surprising because Uniform Cost Search algorithm is a blind search algorithm. Because it does not have specific goal it blindly searches for a goal state whereas A* algorithm with Manhattan distance heuristic function is goal - focused and at each step it gets closer to the goal.

First test: Goal [2,8]

```
graph = [ ['O', 'O', 'O', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
        ['O', 'O', 'O', 'O', 'O', 'O', 'X', 'X', 'X', 'X'],
        ['O', 'O', 'O', 'S', 'O', 'O', 'O', 'O', 'G', 'X'],
        ['X', 'O', 'O', 'S', 'O', 'O', 'O', 'O', 'O', 'O'],
        ['X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'O', 'O'],
        ['X', 'X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'X']
       ]
```

Second Test: Goal [4,7]

```
graph = [ ['O', 'O', 'O', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
        ['O', 'O', 'O', 'O', 'O', 'O', 'X', 'X', 'X', 'X'],
        ['O', 'O', 'O', 'S', 'O', 'O', 'O', 'O', 'O', 'X'],
        ['X', 'O', 'O', 'S', 'O', 'O', 'O', 'O', 'O', 'O'],
        ['X', 'X', 'X', 'X', 'X', 'O', 'O', 'G', 'O', 'O'],
        ['X', 'X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'X']
       ]
```

Third Test: Goal [3,2]

```
graph = [ ['O', 'O', 'O', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
        ['O', 'O', 'O', 'O', 'O', 'O', 'X', 'X', 'X', 'X'],
        ['O', 'O', 'O', 'S', 'O', 'O', 'O', 'O', 'O', 'X'],
        ['X', 'O', 'G', 'S', 'O', 'O', 'O', 'O', 'O', 'O'],
        ['X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'O', 'O'],
        ['X', 'X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'X']
       ]
```

Fourth Test: Goal [0,0]

```
graph = [ ['G', 'O', 'O', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],
        ['O', 'O', 'O', 'O', 'O', 'O', 'X', 'X', 'X', 'X'],
        ['O', 'O', 'O', 'S', 'O', 'O', 'O', 'O', 'O', 'X'],
        ['X', 'O', 'O', 'S', 'O', 'O', 'O', 'O', 'O', 'O'],
        ['X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'O', 'O'],
        ['X', 'X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'X']
       ]
```

Fifth Test: Goal [1,5]

graph = [['O', 'O', 'O', 'X', 'X', 'X', 'X', 'X', 'X', 'X'],

['O', 'O', 'O', 'O', 'O', 'G', 'X', 'X', 'X', 'X'],

['O', 'O', 'O', 'S', 'O', 'O', 'O', 'O', 'O', 'X'],

['X', 'O', 'O', 'S', 'O', 'O', 'O', 'O', 'O', 'O'],

['X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'O', 'O'],

['X', 'X', 'X', 'X', 'X', 'X', 'O', 'O', 'O', 'X']

]

Tests	UCS memory (Successor function calls)	UCS time (seconds)	A* memory (Successor function calls)	A* time (seconds)
Test1	122 times	0.1818013	28 times	0.1125811
Test2	46 times	0.0194473	8 times	0.003212
Test3	139 times	0.2023223	73 times	0.2335319
Test4	41 times	0.0116313	6 times	0.040105
Test5	12 times	0.0045600	6 times	0.0043063