

Assignment 1

Using Informed and Uninformed Search Algorithms to Solve 8-Puzzle

Name: Youssef Mohamed Kishk

ID: 3812

8		6
5	4	7
2	3	1

	1	2
3	4	5
6	7	8

Description:

- An instance of the 8-puzzle game consists of a board holding 8 distinct movable tiles, plus an empty space. For any such board, the empty space may be legally swapped with any tile horizontally or vertically adjacent to it. In this assignment, the blank space is going to be represented with the number 0.
- Given an initial state of the board, the search problem is to find a sequence of moves that transitions this state to the goal state, that is, the configuration with all tiles arranged in ascending order 0,1,2,3,4,5,6,7,8.
- The search space is the set of all possible states reachable from the initial state. The blank space may be swapped with a component in one of the four directions '**Up**', '**Down**', '**Left**', '**Right**', one move at a time.
- It's required to implement the game solution using different algorithms, **BFS**, **DFS**, **A*** (Manhattan Distance & Euclidean Distance).

○ Example:

<i>parent</i> =	1	2	5
	3	4	
	6	7	8

 \Rightarrow

<i>child</i> =	1	2	
	3	4	5
	6	7	8

$parent =$	1	2	
	3	4	5
	6	7	8

 \Rightarrow

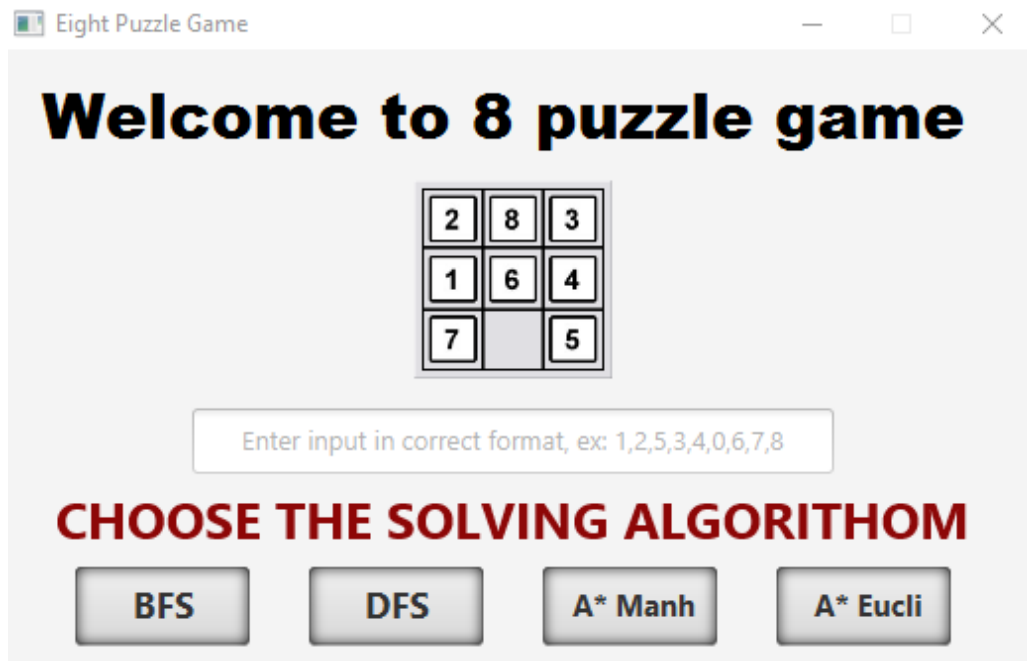
$child =$	1		2
	3	4	5
	6	7	8

<i>parent</i> =	1		2	\Rightarrow	<i>child</i> =		1	2
	3	4	5			3	4	5
	6	7	8			6	7	8

- **Note:** I assume that the input initial case is solvable.

Testing GUI

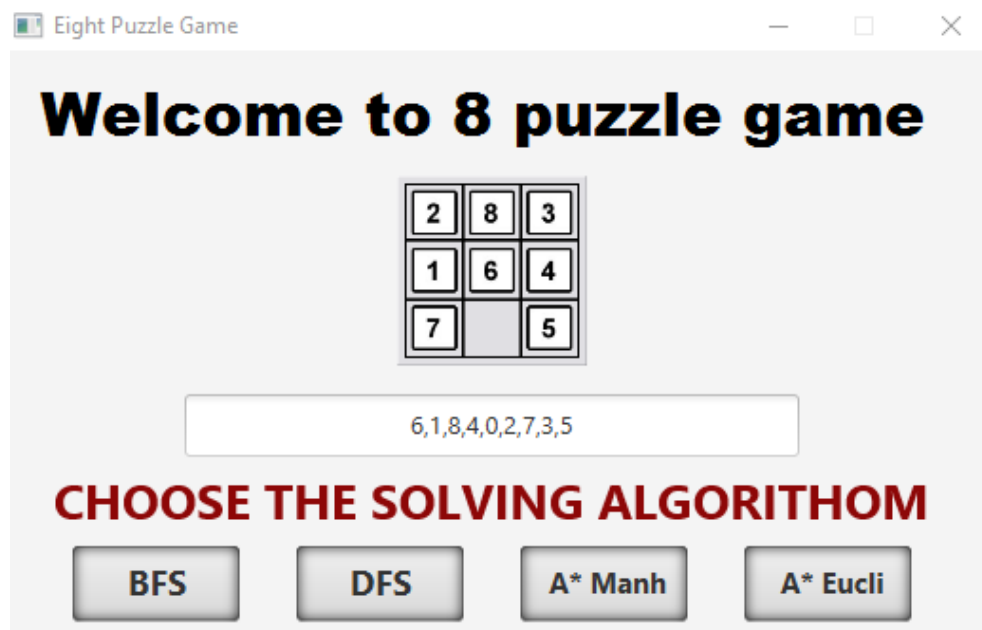
- You will find attached in the project folder an excitable GUI for the game solver (**Eight Puzzle Game**)



Usage Guide

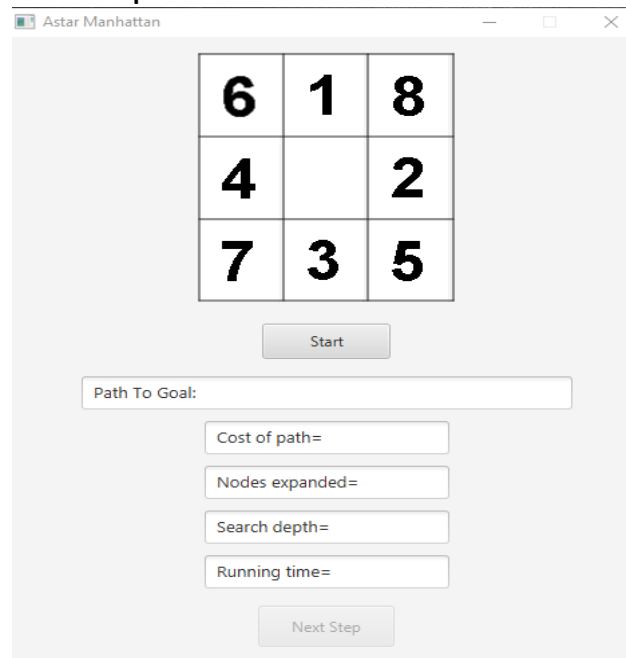
- [1] Enter a solvable example in the format shown

Ex : 6,1,8,4,0,2,7,3,5



[2] Choose the algorithm you want to test (ex : **A* Manh**)

After choosing the algorithm the following screen will appear with the initial state of the puzzle

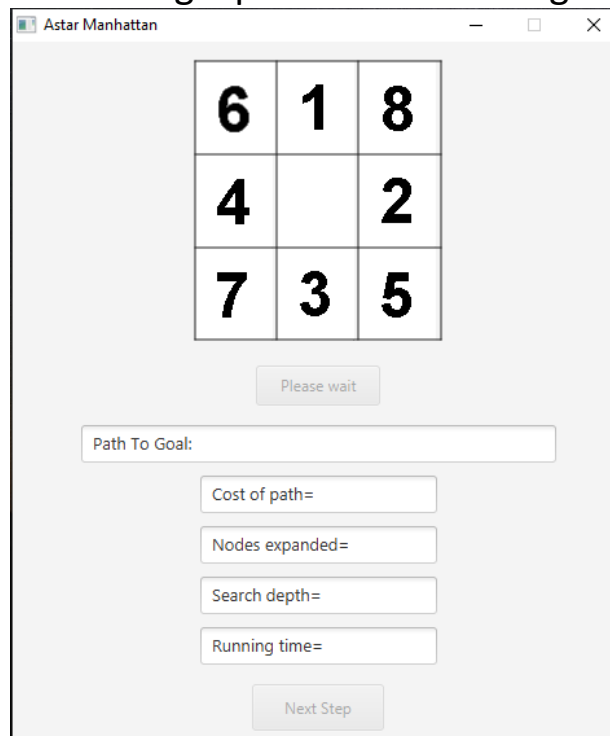


The screenshot shows a window titled "Astar Manhattan". Inside, there is a 3x3 grid representing a puzzle state. The numbers in the grid are:

6	1	8
4		2
7	3	5

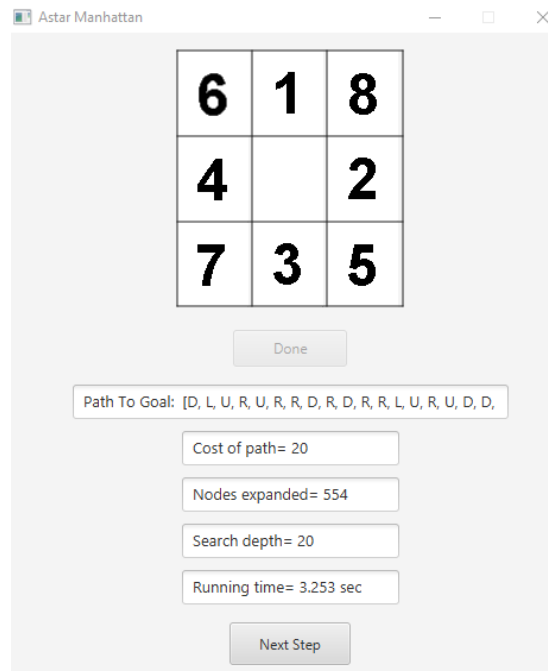
Below the grid is a "Start" button. Underneath that is a text input field labeled "Path To Goal:". Below the input field are four more text input fields, each preceded by a label: "Cost of path=", "Nodes expanded=", "Search depth=", and "Running time=". At the bottom is a "Next Step" button.

[3] Press start and wait for the algorithm to finish solving the puzzle
(It may take some time according to the chosen algorithm, the GUI will freeze showing a please wait message until it finishes)



The screenshot shows the same "Astar Manhattan" window. The 3x3 grid is identical to the previous one. However, the "Start" button is now disabled, and a "Please wait" button has appeared in its place. The other elements (input fields and "Next Step" button) remain the same.

[4] After finishing, the results of the puzzle solving will appear in the shown text fields and also it's printed in an external file in the same path of the GUI.



[5] If you want to check the steps the algorithm passed through to reach the final state keep pressing **next step** button and notice the change in the puzzle.

	1	2
3	4	5
6	7	8

(Puzzle solved)

Implemented Algorithms

Note: **A comparison is presented at the report last page**

1. BFS

- The algorithm idea is to horizontally explore the nodes within the tree until we reach the goal.
- In general its search depth value is fine compared to other algorithms as DFS.
- The main defect is the long running time because BFS is an uninformed search so we have no information about the path that may help us reach the goal.
- BFS is implemented using Queue data structure.

• **Algorithm:**

BFS search

```
function BREADTH-FIRST-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE :

    frontier = Queue.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.dequeue()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.enqueue(neighbor)

    return FAILURE
```

- Test case: I/P: 6,1,8,4,0,2,7,3,5

BFS

6	1	8
4		2
7	3	5

Done

Path To Goal: [U, R, D, L, L, U, R, R, D, L, L, U, R, R, D, L, L, D, R,

Cost of path= 20

Nodes expanded= 54094

Search depth= 20

Running time= 146.518 sec

Next Step

2. DFS

- The algorithm idea is to deeply explore the nodes within the tree until we reach the goal.
- In general its search depth value is bad due to searching until end of depth of each node in tree.
- Same as BFS the main defect is the long running time because DFS is an uninformed search so we have no information about the path that may help us reach the goal.
- DFS is implemented using Stack data structure.

- Algorithm:

DFS search

```

function DEPTH-FIRST-SEARCH(initialState, goalTest)
  returns SUCCESS or FAILURE :

  frontier = Stack.new(initialState)
  explored = Set.new()

  while not frontier.isEmpty():
    state = frontier.pop()
    explored.add(state)

    if goalTest(state):
      return SUCCESS(state)

    for neighbor in state.neighbors():
      if neighbor not in frontier  $\cup$  explored:
        frontier.push(neighbor)

  return FAILURE
  
```

Taken from the edX course ColumbiaX: CSMM.101x Artificial Intelligence (AI)

- Test case: I/P: 6,1,8,4,0,2,7,3,5

DFS

6	1	8
4		2
7	3	5

Done

Path To Goal: [U, U, L, D, D, R, U, U, L, D, D, R, U, U, L, D, D, R,

Cost of path= 46142

Nodes expanded= 51015

Search depth= 46142

Running time= 148.378 sec

Next Step

3. A*

- The algorithm idea is to work on finding the goal with putting into consideration the Heuristics that would help reaching the goal in shortest path.
- A* is an informed search method, so we have some knowledge (Heuristics) that lead us to reach the goal in the shortest and fastest way so its running time is a lot better than any uninformed search method.
- A* is implemented using priority queue data structure.
- For the Heuristics we compute it using any of the following 2 methods (**Both are implemented**):
 - Manhattan Distance:
 - It is the sum of absolute values of difference's in the goal's x and y coordinates and the current cell's x and y coordinates respectively.
 - Euclidean Distance:
 - It is the distance between the current cell and the goal cell using the distance formula.

- Algorithm:

A* search

Taken from the edX course ColumbiaX: CSMM.101x Artificial Intelligence (AI)

```
function A-STAR-SEARCH(initialState, goalTest)
    returns SUCCESS or FAILURE : /* Cost  $f(n) = g(n) + h(n)$  */

    frontier = Heap.new(initialState)
    explored = Set.new()

    while not frontier.isEmpty():
        state = frontier.deleteMin()
        explored.add(state)

        if goalTest(state):
            return SUCCESS(state)

        for neighbor in state.neighbors():
            if neighbor not in frontier ∪ explored:
                frontier.insert(neighbor)
            else if neighbor in frontier:
                frontier.decreaseKey(neighbor)

    return FAILURE
```

- Test case:

A* Manhattan
I/P: 6,1,8,4,0,2,7,3,5

Astar Manhattan

6	1	8
4		2
7	3	5

Done

Path To Goal: [D, L, U, R, U, R, R, D, R, D, R, L, U, R, U, D, D,

Cost of path= 20

Nodes expanded= 554

Search depth= 20

Running time= 2.944 sec

Next Step

A* Euclidean

I/P: 6,1,8,4,0,2,7,3,5

Astar Euclidean

6	1	8
4		2
7	3	5

Done

Path To Goal: [L, L, D, R, D, R, U, L, R, L, D, L, R, D, U, U, U, L, U

Cost of path= 20

Nodes expanded= 937

Search depth= 20

Running time= 18.643 sec

Next Step

Conclusion:**For the test case: 6,1,8,4,0,2,7,3,5**

Search algorithm	Cost of path	Nodes expanded	Running time/sec
BFS	20	54094	146.5
DFS	46142	51015	148.3
A* Manhattan	20	554	2.9
A* Euclidean	20	937	18.6

- ✓ As a conclusion, if we reviewed the above table we can notice the following:
1. The cost path is close in all algorithms except for DFS it has very large depth value
 2. In both uninformed search algorithms (BFS, DFS), the number of expanded nodes is too large, while for informed search algorithm (A*) the nodes expanded is much smaller, and that's why Heuristics are powerful.
 3. The running time truly shows how informed search is much better than uninformed search, as the time difference between BFS,DFS (that are close in running time), and A* search is great, so that's why A* is the best algorithm to solve this game as it expand least nodes and reach the goal in the fastest way.
- ✓ Finally concerning the different Heuristics calculating ways, in most cases Manhattan is faster and expands less nodes to reach goal.
-