Intro to Artificial Intelligence

Lab1 : 8-Puzzle game

TEAM MEMBERS

Name	ID
Nayra Ibrahim Ahmed Mohamed	21011504
Ibrahem Mohamed El-Sayed	21010023
Mohamed Mohamed Mohamed Abd-ELmoneim	21011213

PROBLEM STATEMENT

You will need to implement the 8 puzzle search problem using the four search algorithms:

- BFS
- DFS
- Iterative DFS
- A*

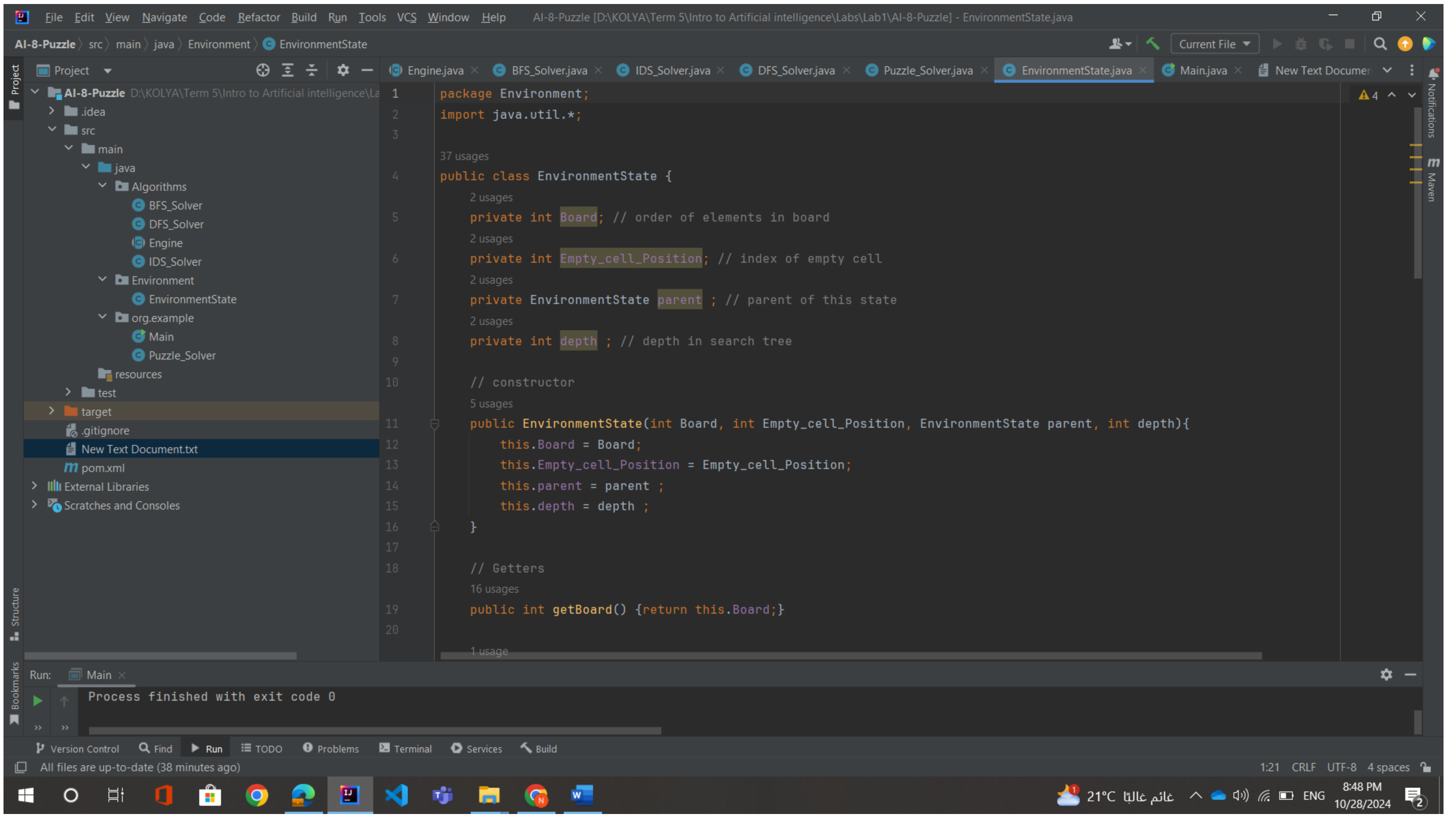
the report should contain the data structure used (if any) and the explanation for each algorithm, Assumptions and details you find them necessary to be clarified, Any extra work and Sample runs. You should show your algorithm and how it operates.

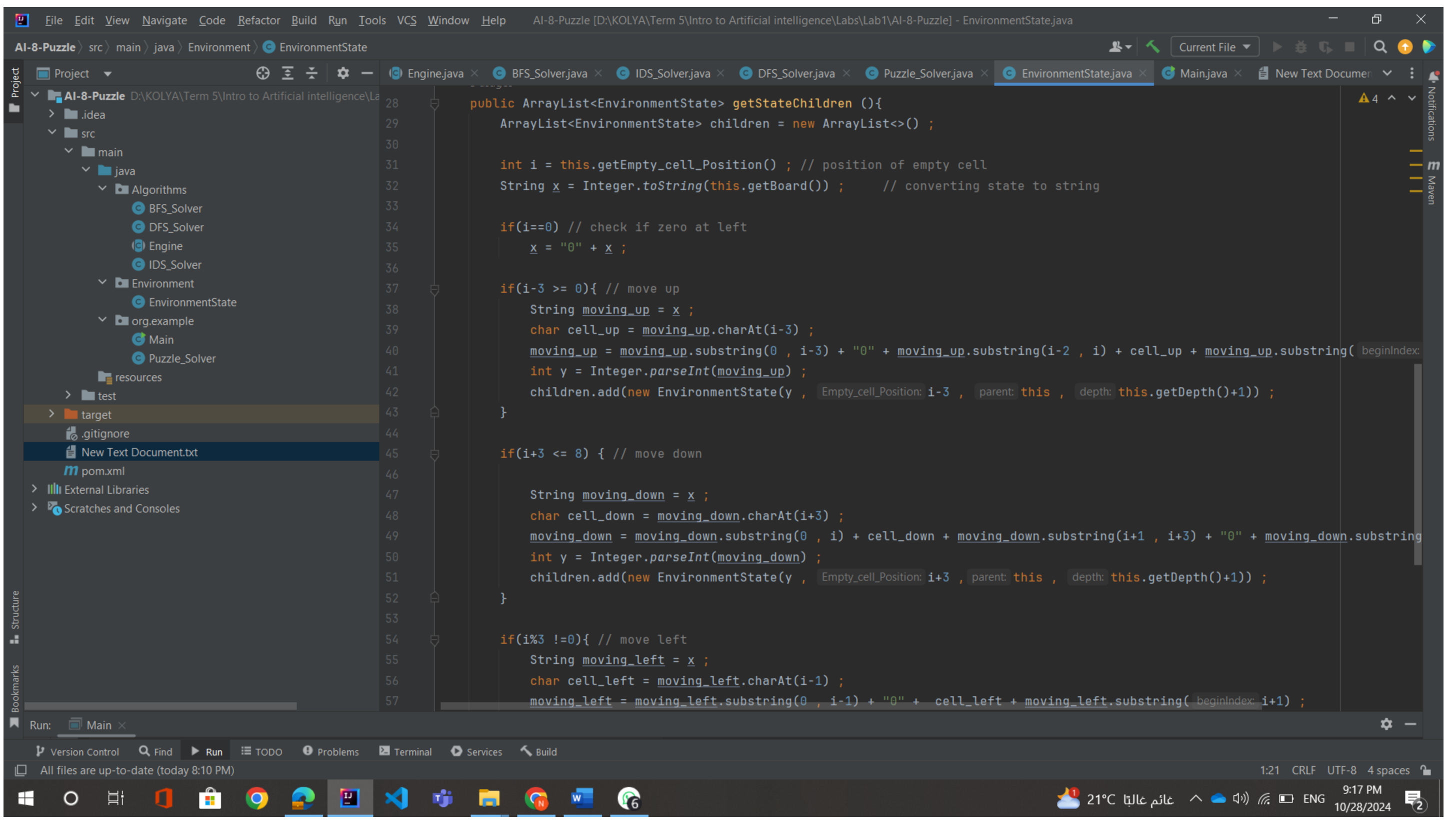
1. BFS

Data Structures :

1. We define a data Structure called **EnvironmentState** that contains a collection of some information about the puzzle at each step that will be used as node for the search tree for (BFS , DFS , IDS Algorithms).
   It contains : Integer represent the board , integer represent the index of the empty cell instead of searching for empty cell every time we expand a node , depth of this node at the search tree , parent node of type EnvironmentState that links each node with the node created it to get the path of the game by backtracking.

   This data structure return the all possible children of a node (up – down – left – right)
   At this order.


AI-8-Puzzle [D:\KOLYA\Term 5\Intro to Artificial intelligence\Labs\Lab1\AI-8-Puzzle] - EnvironmentState.java
package Environment;
import java.util.*;
37 usages
public class EnvironmentState {
2 usages
private int Board; // order of elements in board
2 usages
private int Empty_cell_Position; // index of empty cell
2 usages
private EnvironmentState parent ; // parent of this state
2 usages
private int depth ; // depth in search tree
// constructor
5 usages
public EnvironmentState(int Board, int Empty_cell_Position, EnvironmentState parent, int depth){
this.Board = Board;
this.Empty_cell_Position = Empty_cell_Position;
this.parent = parent ;
this.depth = depth ;
}
// Getters
16 usages
public int getBoard() {return this.Board;}
Process finished with exit code 0
All files are up-to-date (38 minutes ago)
8:48 PM
10/28/2024


AI-8-Puzzle [D:\KOLYA\Term 5\Intro to Artificial intelligence\Labs\Lab1\AI-8-Puzzle] - EnvironmentState.java
public ArrayList<EnvironmentState> getStateChildren (){
ArrayList<EnvironmentState> children = new ArrayList<>() ;
int i = this.getEmpty_cell_Position() ; // position of empty cell
String x = Integer.toString(this.getBoard()) ; // converting state to string
if(i==0) // check if zero at left
x = "0" + x ;
if(i-3 >= 0){ // move up
String moving_up = x ;
char cell_up = moving_up.charAt(i-3) ;
moving_up = moving_up.substring(0 , i-3) + "0" + moving_up.substring(i-2 , i) + cell_up + moving_up.substring(beginIndex:
int y = Integer.parseInt(moving_up) ;
children.add(new EnvironmentState(y , Empty_cell_Position: i-3 , parent: this , depth: this.getDepth()+1)) ;
}
if(i+3 <= 8) { // move down
String moving_down = x ;
char cell_down = moving_down.charAt(i+3) ;
moving_down = moving_down.substring(0 , i) + cell_down + moving_down.substring(i+1 , i+3) + "0" + moving_down.substring
int y = Integer.parseInt(moving_down) ;
children.add(new EnvironmentState(y , Empty_cell_Position: i+3 , parent: this , depth: this.getDepth()+1)) ;
}
if(i%3 !=0){ // move left
String moving_left = x ;
char cell_left = moving_left.charAt(i-1) ;
moving_left = moving_left.substring(0 , i-1) + "0" + cell_left + moving_left.substring(beginIndex: i+1) ;
All files are up-to-date (today 8:10 PM)
9:17 PM
10/28/2024

2. We also used **[Queue]** for frontier list that makes us expand level by level.
3. We also used **[Hashset]** that contains all nodes that were visited or nodes that are in the frontier at the current time . Hashset gives O(1) operations (instead of linear search in visited and frontier queues) .we now don't want visited array.

Explanation of the algorithm:

Expand shallowest node

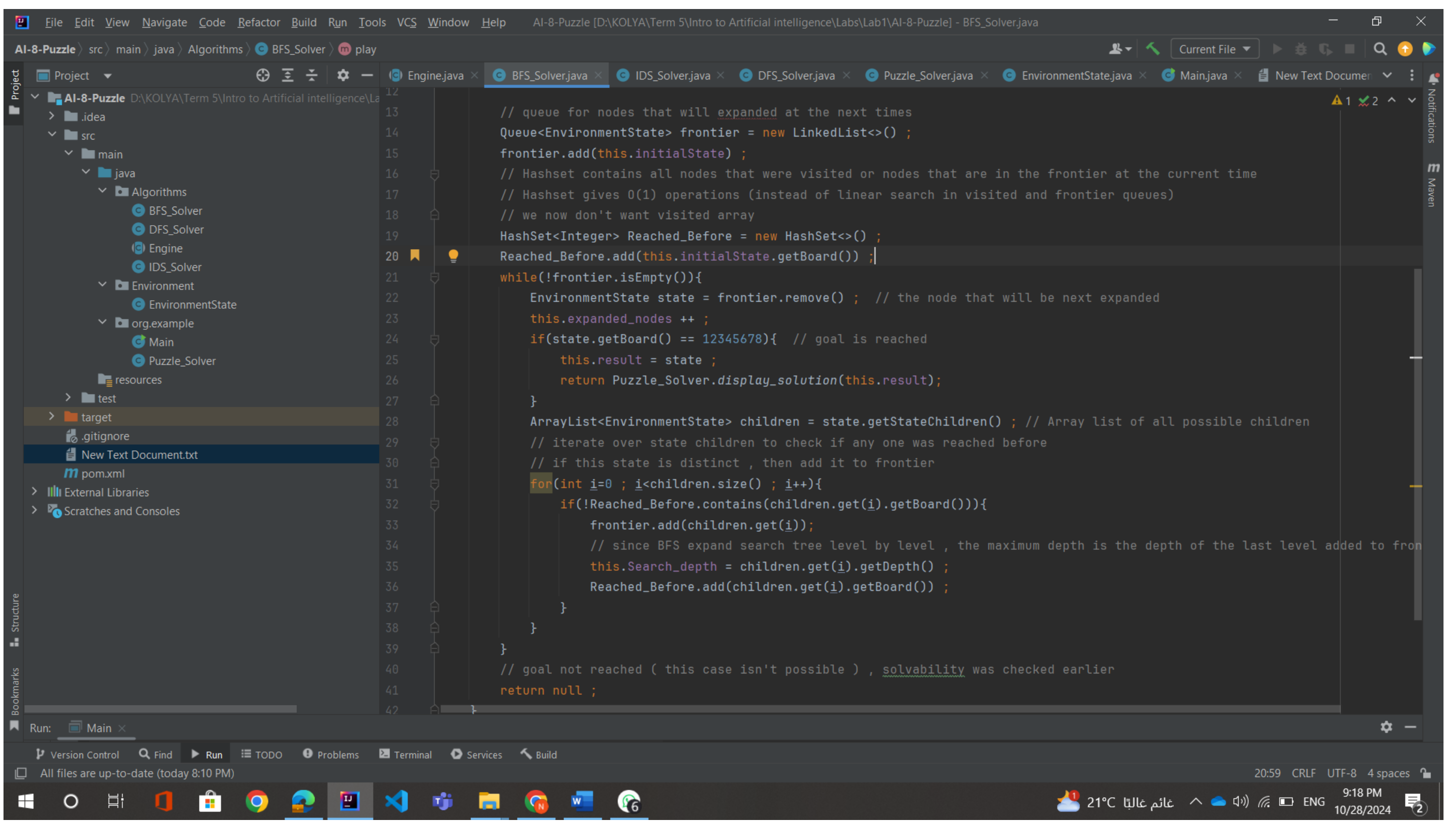
The algorithm takes the state of the puzzle , add it to the frontier queue.

Iterate over the frontier unless it is empty.

Remove the top of the frontier , check if it is the goal return it ,

If not the goal compute all possible children (up – down – left – right) ,

Check every child if it is expanded before then ignore it , if no add it to the frontier then repeat.



Assumptions

The puzzle has solution at finite depth.

2. DFS

Data Structures :

1. Environment state as we mentioned at BFS algorithm.
2. We also used Stack for frontier list that makes us expand deepest first.
3. We also used **Hashset** that contains all nodes that were visited or nodes that are in the frontier at the current time . Hashset gives O(1) operations (instead of linear search in visited and frontier queues) .we now don't want visited array.

Explanation of the algorithm:

Expand deepest first.

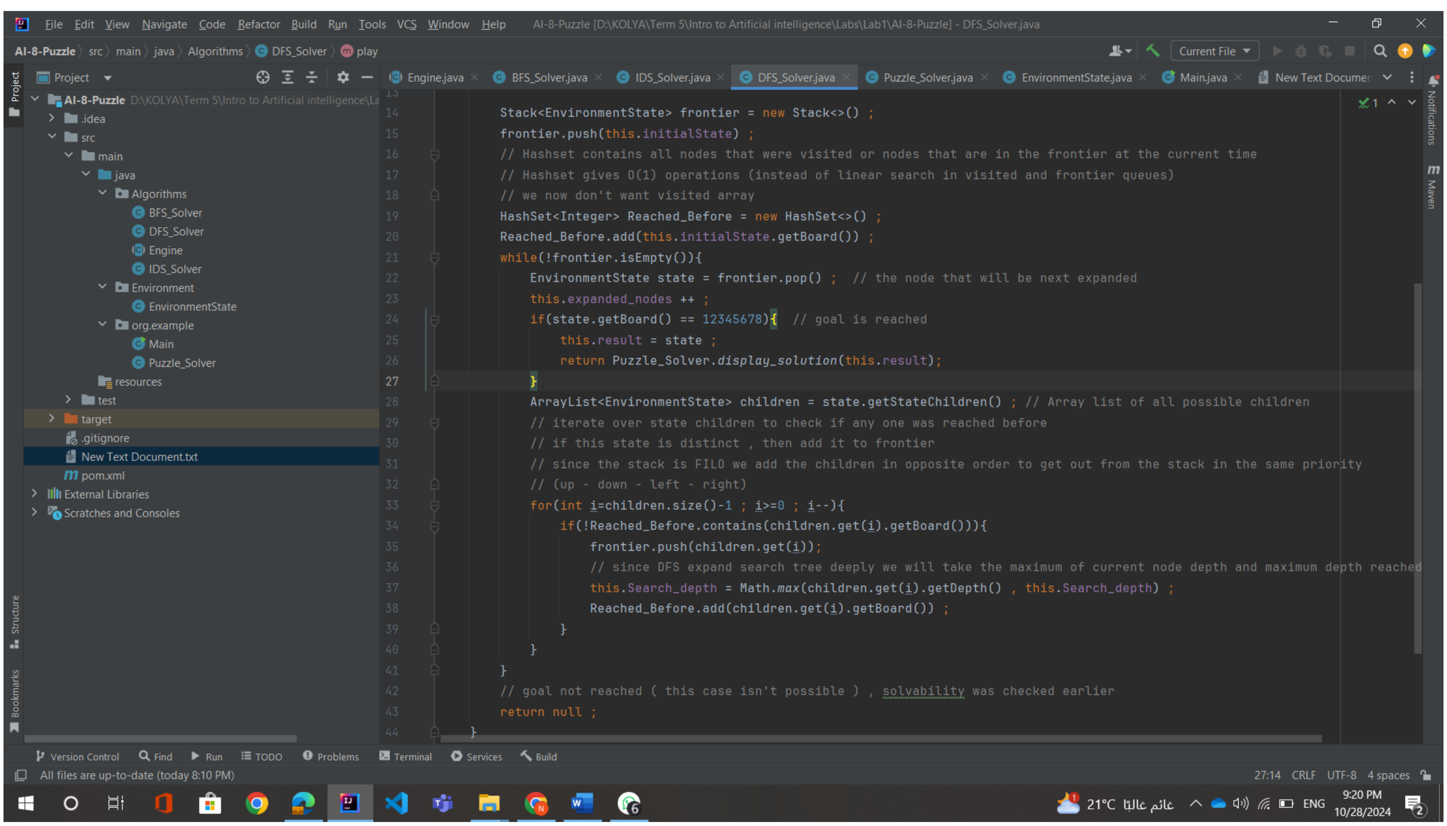
The algorithm takes the state of the puzzle , add it to the frontier stack.

Iterate over the frontier unless it is empty.

Remove the top of the Stack , check if it is the goal return it ,

If not the goal compute all possible children (up – down – left – right) ,

Check every child if it is expanded before then ignore it , if no add it to the frontier then repeat. Check deepest first if not the goal , back to lower depth then continue expansion.



Assumptions

The puzzle has solution at finite depth.

3.IDS

Data Structures:

1. Environment state as we mentioned at BFS algorithm.
2. We also used Stack for frontier list that makes us expand deepest first.
3. We also used **HashMap** that contains all nodes that were visited or nodes that are in the frontier at the current time . HashMap gives O(1) operations (instead of linear search in visited and frontier queues) .we now don't want visited array. HashMap contains the reached states and their depth.
   HashMap <key , value> = HashMap<board , depth > because board is unique so it is the key.

Explanation of the algorithm:

Combines the benefits of BFS and DFS.

Idea: Iteratively increase the search limit until the depth of the shallowest solution d is reached.

Applies DLS with increasing limits.

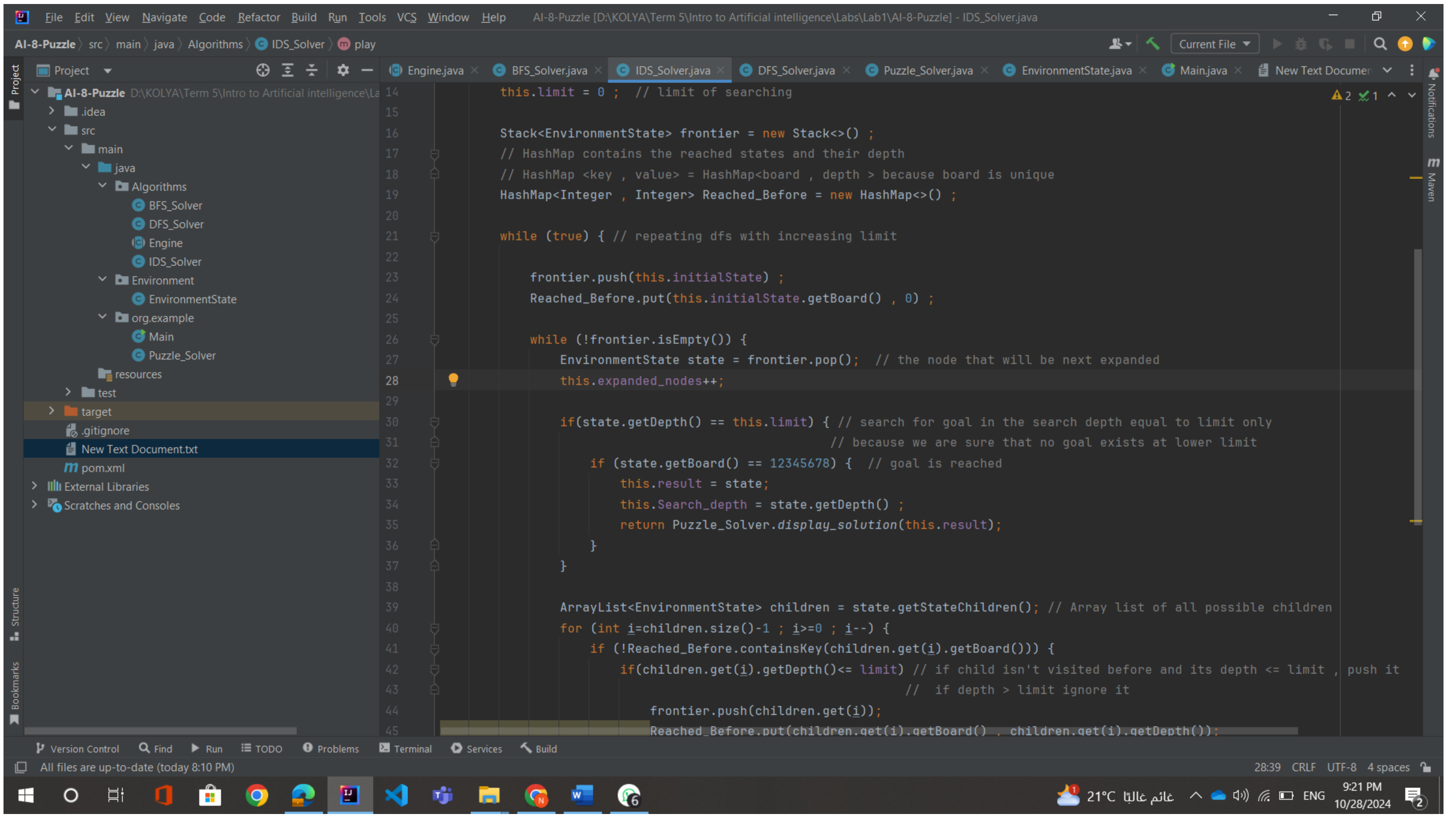
The algorithm takes the state of the puzzle , add it to the frontier Stack.

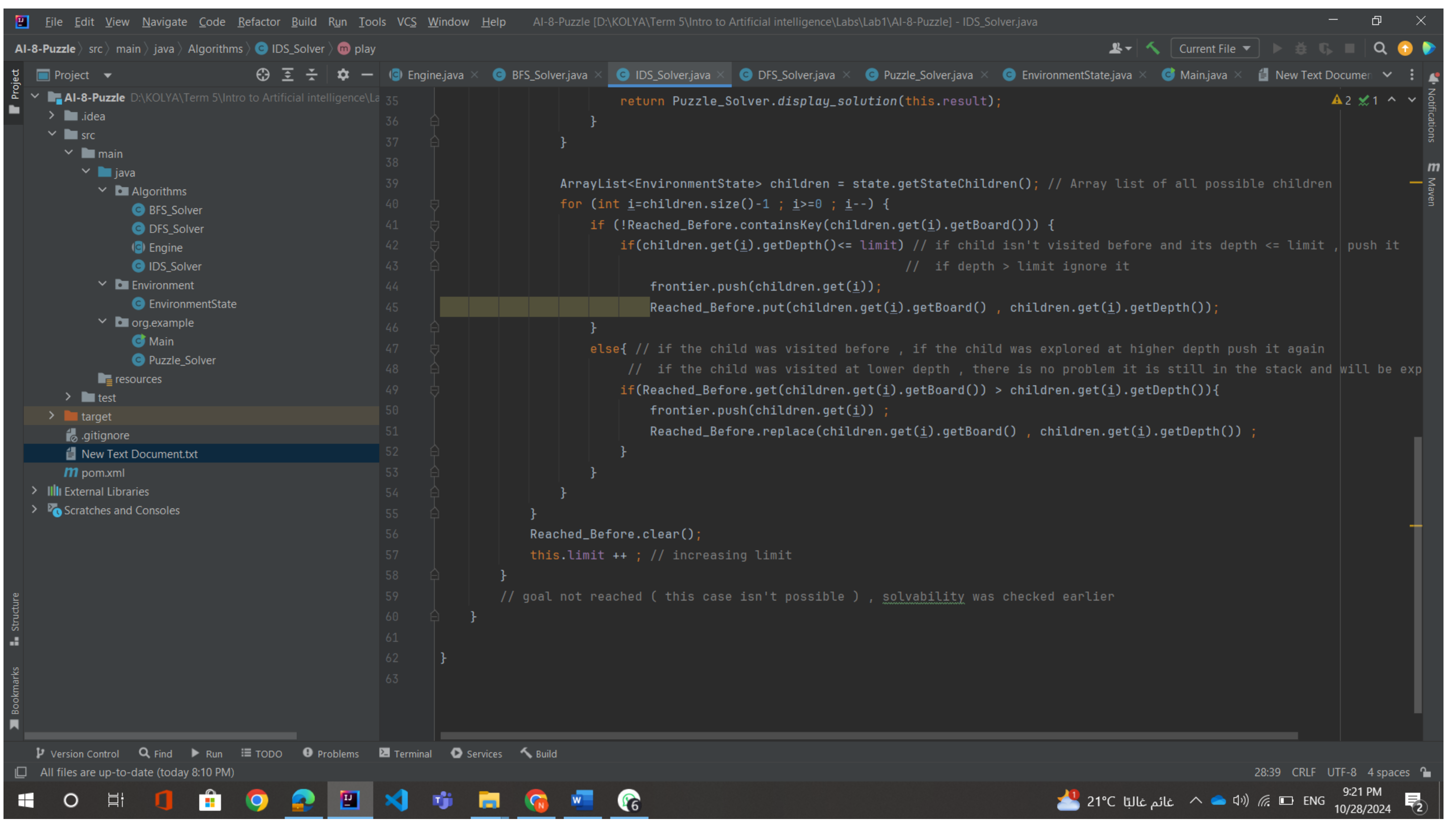
Iterate over the frontier unless it is empty.

Remove the top of the frontier , if the node has depth equal to the limit of the search check if it is the goal (because we are sure that no goal exists at lower limit) if yes return it ,

If not the goal compute all possible children (up – down – left – right) ,

Check every child if it is expanded before check if current depth is less than its last depth then add it if no ignore it , if no compare child depth with the limit if the depth > limit ignore it , if no add it to the frontier then repeat.





Assumptions

The puzzle has solution at finite depth.

4. A*

Way of storing states:

State: <Long> [stored in **64bits** in java]

Cost	Board	Parent	Direction
2 digits	9 digits	4 digits	1 digit

○ Cost: $g(n) + h(n)$;

Worst case for $h(n)$: =

4	3	4
3	2	3
4	3	4

4(4) + 4(3) + 2 = 30 [according to Manhattan]

Worst case for $g(n) = \max\,(depth) =\ 31$ [proved by exhaustive search methods]

∴ $g(n)$ = 6; thus $f(n) = 30 + 31 < 61$ → **2 digits**.

Notice that: this estimation is impossible since as we get nearer to the goal $h(n)$ decreases, but this is just for the sake of number of digits.

○ Board: It has 9 slots; thus **9 digits**.

○ Parent: In case of Full tree (worst case) number of inner nodes (parents) $< N$;

Where N is the total number of nodes in the tree which is equal to number of permutations of the empty slot:

2	3	2
3	4	3
2	3	2

$= 2^4 * 3^4 * 4 = 5184$ = N

∴ Maximum number of parents < 5184 → needs **4 digits**.

So the hash map key of the state's parent that consists of 4 digits which is created randomly using `AStarSearch.createId()` function is stored in the 4 digits of the parent.

○ Direction: Is one of the following four: root (0) [came from no movement], right (1), left (2), up (3), down (4) → needs **1 digit**.

Data structures used:

Visited List: Hash Map.

Frontier List: Priority Queue.

Path to goal is traced using: Stack.

Explanation of the algorithm:

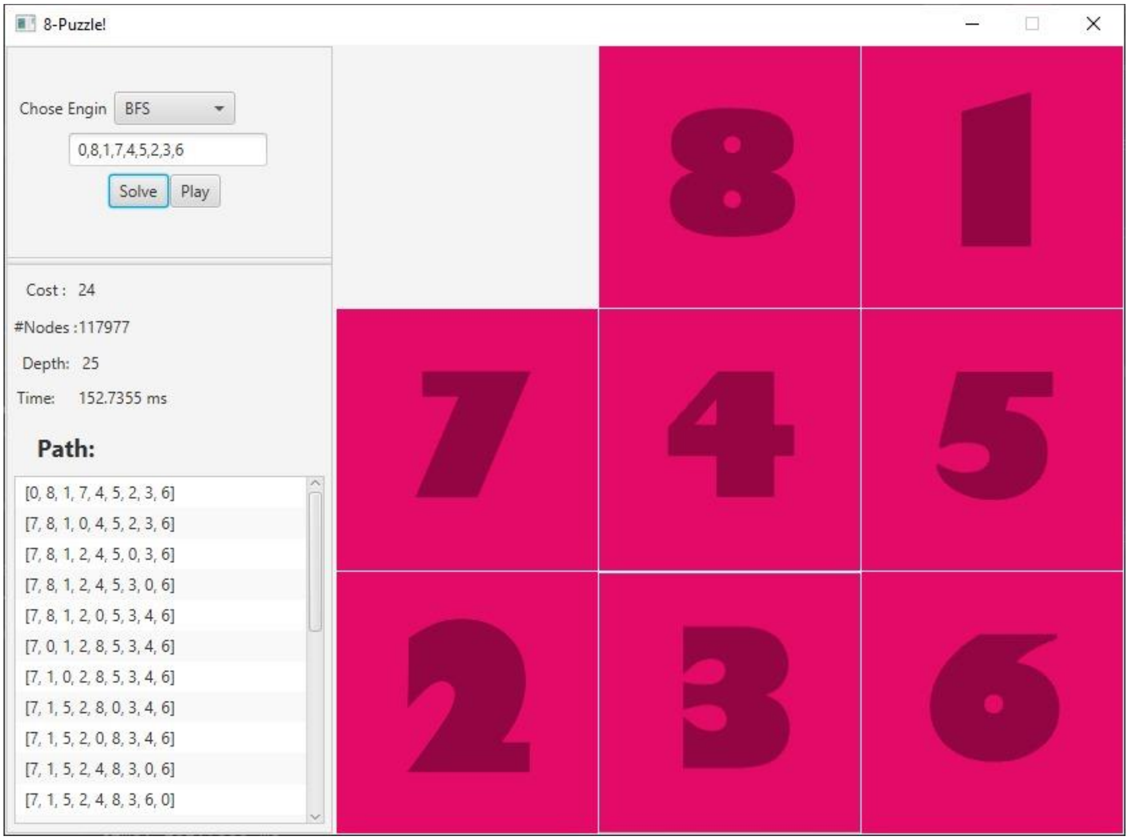
New states expanded are popped from the frontier and added as visited, and its new children are put in the frontier in its correct order with respect to the cost of it, which is calculated as the addition of the depth of the current state and the estimation to the goal (heuristic $f(x)$) (with Manhattan or Euclidean) $(g(x) + h(x))$.

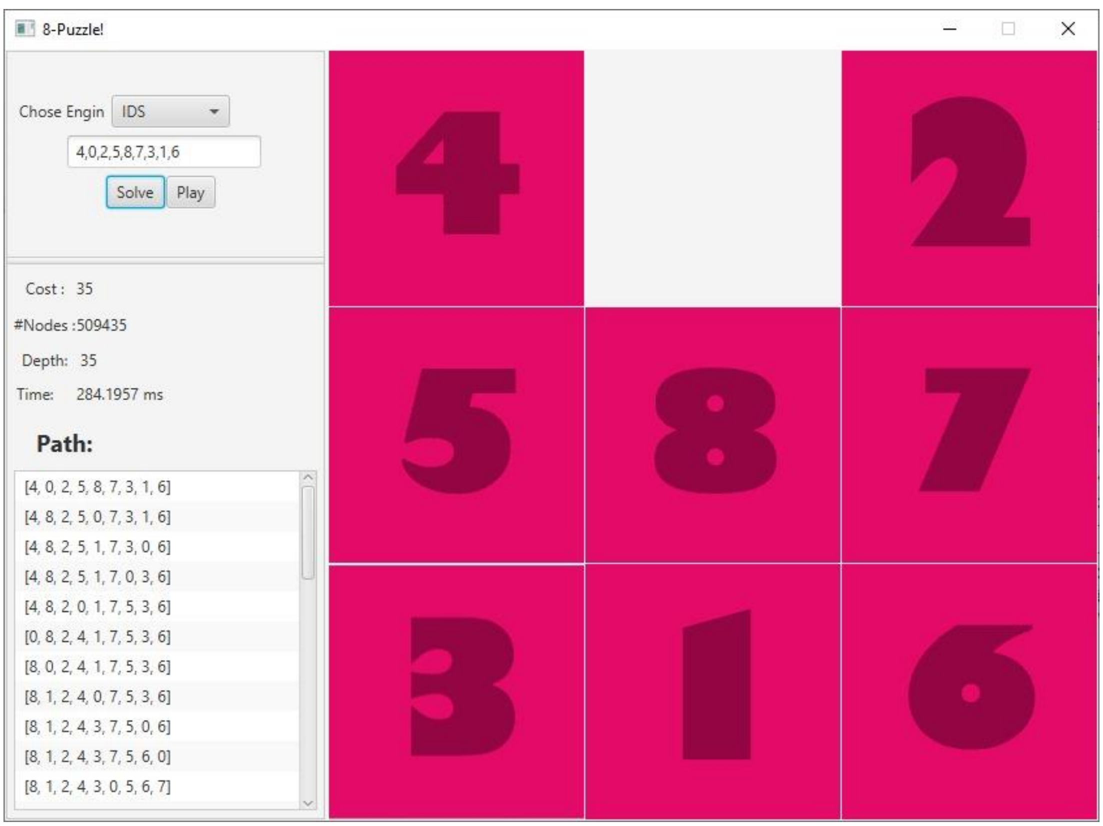
Comparison between the two heuristics:

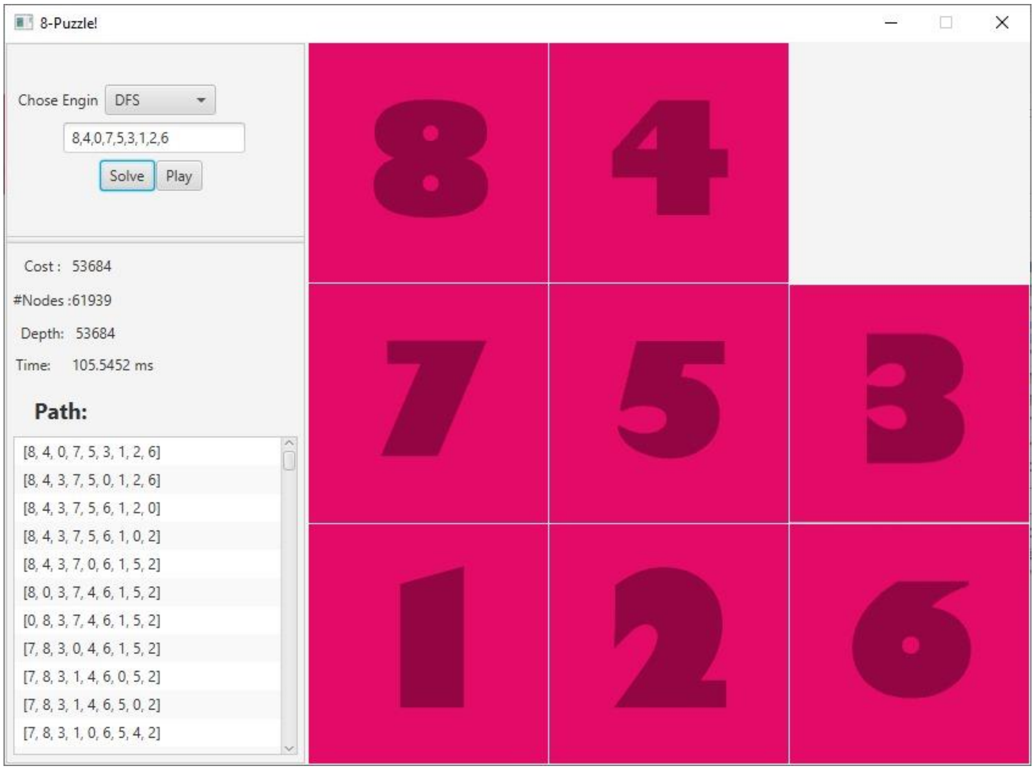
Manhattan is **more admissible**, because it gets the sum of horizontal and vertical movements to the goal which is more accurate than Euclidean method, which is calculating the straight distance between the current state and the goal.

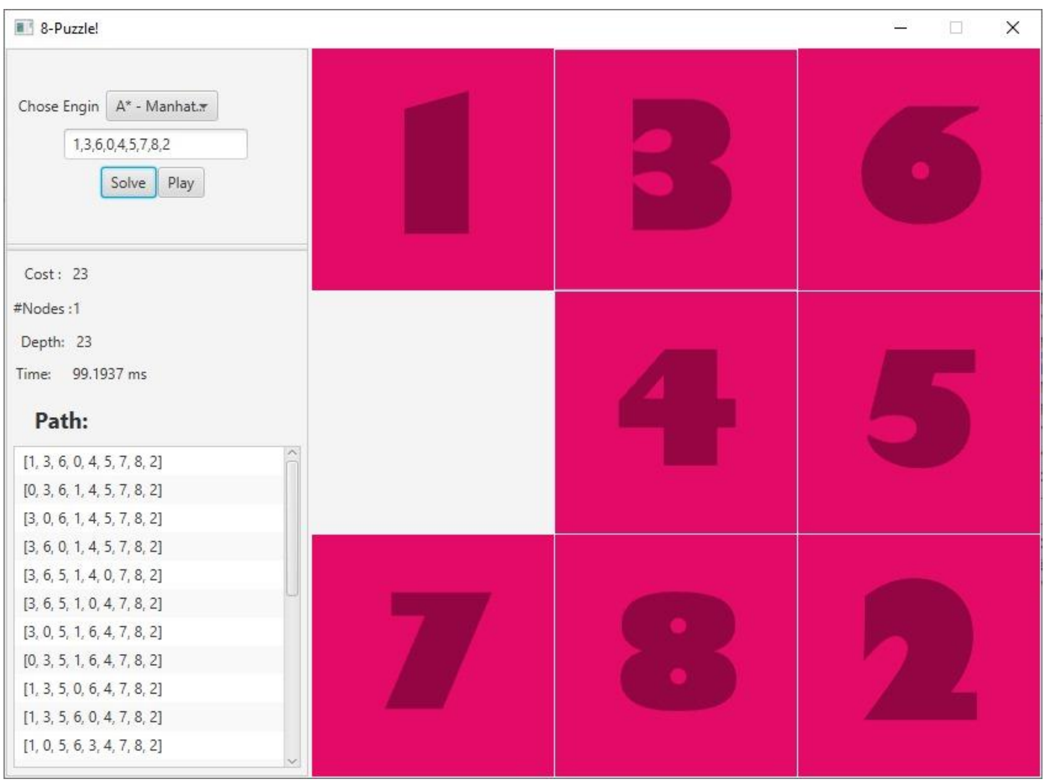
	Initial state	Manhattan	Euclidean
# of expanded nodes	5 1 2 3 4 0 6 7 8	32 nodes	59 nodes
Output path		[up, left, left, down, right, up, right, down, left, left, up]	[up, left, left, down, right, up, right, down, left, left, up]
Goal depth		11	11
Running time		14 msec	18 msec
# of expanded nodes	1 2 3 4 5 0 6 7 8	125 nodes	209 nodes
Output path		[left, left, up, right, right, down, left, up, left, down, right, right, up, left, left]	[left, left, up, right, right, down, left, up, left, down, right, right, up, left, left]
Goal depth		15	15
Running time		24 msec	33 msec
# of expanded nodes	8 6 7 2 5 4 3 0 1	2526 nodes	9684 nodes
Output path		[up, right, down, left, up, up, left, down, down, right, up, up, left, down, down, right, up, right, up, left, down, right, down, left, up, left, up]	[right, up, left, down, down, right, up, right, up, left, down, right, down, left, up, left, up, right, down, left, down, right, up, up, left]
Goal depth		27	25
Running time		389 msec	4176 msec

TEST CASES BY THE GUI:






8-Puzzle!
Chose Engin
DFS
8,4,0,7,5,3,1,2,6
Solve
Play
Cost : 53684
#Nodes :61939
Depth: 53684
Time: 105.5452 ms
Path:
[8, 4, 0, 7, 5, 3, 1, 2, 6]
[8, 4, 3, 7, 5, 0, 1, 2, 6]
[8, 4, 3, 7, 5, 6, 1, 2, 0]
[8, 4, 3, 7, 5, 6, 1, 0, 2]
[8, 4, 3, 7, 0, 6, 1, 5, 2]
[8, 0, 3, 7, 4, 6, 1, 5, 2]
[0, 8, 3, 7, 4, 6, 1, 5, 2]
[7, 8, 3, 0, 4, 6, 1, 5, 2]
[7, 8, 3, 1, 4, 6, 0, 5, 2]
[7, 8, 3, 1, 4, 6, 5, 0, 2]
[7, 8, 3, 1, 0, 6, 5, 4, 2]
8
4
7
5
3
1
2
6


8-Puzzle!
Chose Engin
A* - Manhat
1,3,6,0,4,5,7,8,2
Solve
Play
Cost : 23
#Nodes :1
Depth: 23
Time: 99.1937 ms
Path:
[1, 3, 6, 0, 4, 5, 7, 8, 2]
[0, 3, 6, 1, 4, 5, 7, 8, 2]
[3, 0, 6, 1, 4, 5, 7, 8, 2]
[3, 6, 0, 1, 4, 5, 7, 8, 2]
[3, 6, 5, 1, 4, 0, 7, 8, 2]
[3, 6, 5, 1, 0, 4, 7, 8, 2]
[3, 0, 5, 1, 6, 4, 7, 8, 2]
[0, 3, 5, 1, 6, 4, 7, 8, 2]
[1, 3, 5, 0, 6, 4, 7, 8, 2]
[1, 3, 5, 6, 0, 4, 7, 8, 2]
[1, 0, 5, 6, 3, 4, 7, 8, 2]
1
3
6
4
5
7
8
2