Department of Computer Engineering

CS441

Introduction to Artificial Intelligence

Fall, 2019

Project Report

Title

Implementation of A Search Algorithm* and *Dijkstra's Algorithm*

Group Members:

190201038 – Ibrahim Berber

160202001 – Muharrem Kilinc

1. Introduction

Artificial Intelligence (or AI), is the study of building agents that act rationally, and these agents perform *search algorithms* to achieve the task.

Bread-First Search and *Depth-First Search* are in *Blind* category, and *A* search*, *Greedy Search* and *Best-First Search* are in *Heuristic* category.

In this project, we have worked on the illustration of two algorithms that has been thought in the CS441 – Introduction to Artificial Intelligence course:

- A* Search Algorithm
- Dijkstra's Search Algorithm

2. Description of Algorithms

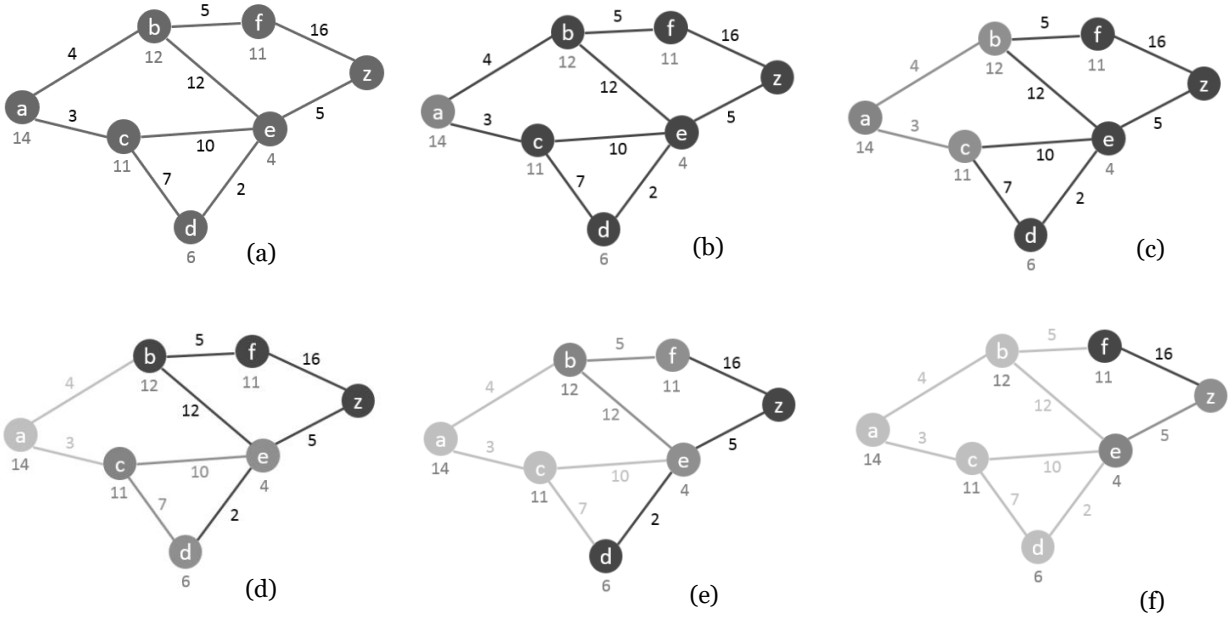
a. A* Search Algorithm

One of the graph search algorithms is *A* Search Algorithm* and it is responsible for finding a path that connects given initial node and a specified destination node. It leverages it power from *heuristic estimate*, which is an estimation of the best route that goes through that node. A* Search Algorithm takes the information of *Best-First Search*, that is selecting the nodes that are close to the destination, and the information of *Dijkstra's Search Algorithm*, that is selecting the nodes that are close to the starting node. Having those two-information combined, we got the formula shown in Equation 1, as follows:

$$f(n) = g(n) + h(n) \qquad \text{[Equation 1]}$$

where,

$g(n)$: the exact cost of the path from the starting point to any vertex (node) n
$h(n)$: heuristic estimated cost from vertex (node) n to the goal



Execution of A* Algorithm

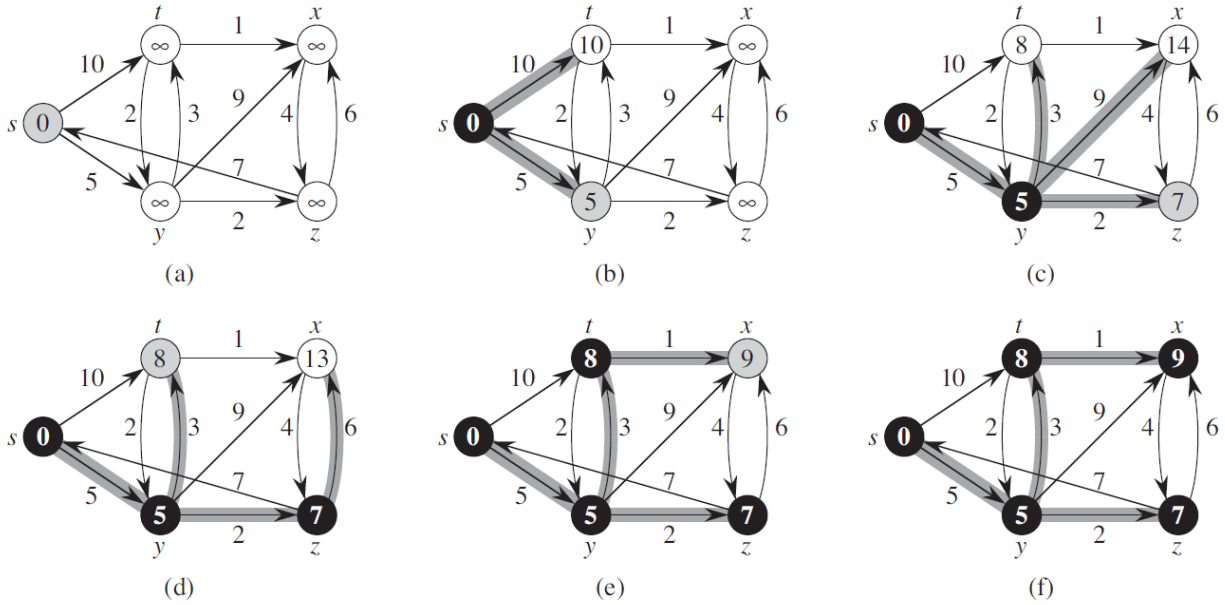
b. Dijkstra's Search Algorithm

In Dijkstra's Search Algorithm, shortest distances of nodes n from the source are kept in a data-structure, called *array*. From initial node, the shortest distance is to itself is *zero*. Distance to all other nodes are *infinity*, as we have not processed those nodes yet. After having processed all the nodes, the *array* will have the shortest distance of node from the initial source to every available node. Finding all distances for each node, in the end, yields a shortest path from the beginning to destination.

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

```
RELAX(u, v, w)
1  if v.d > u.d + w(u, v)
2      v.d = u.d + w(u, v)
3      v.π = u
```

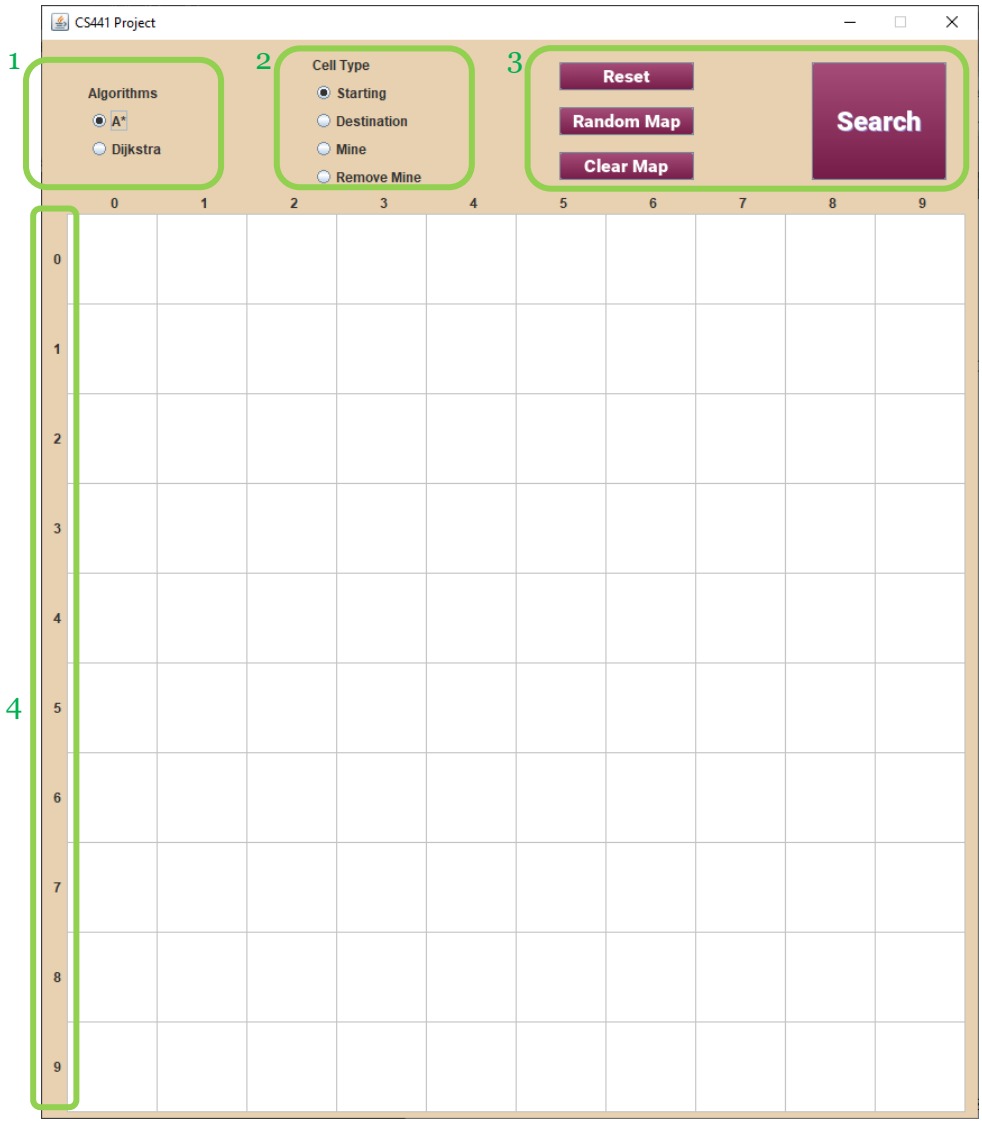
Pseudocode of Dijkstra's Algorithm



Execution of Dijkstra's Algorithm

3. Implementation

The algorithms introduced in Part 2, are attempted to implement in *Java Programming Language*, as it has the features of performant, less-complexity, support of object-orientation and GUI libraries (*swing* and *awt)*.



Java Application Overview

1: *Radio Button* – selecting the Algorithm to be performed.

2: *Radio Button* – changing type of the cells.

3: *Buttons* – which performs specific tasks that are assigned to them.

4: *Position Labels* – coordinates system illustration

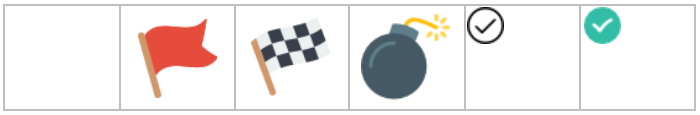
a. Classes

Path.java contains the following classes:

- **`class Path -`** Positioning the GUI elements, declaration of essential variables such as *frame size*, *number of cells, selection of algorithms,* etc.
- **`class Map –`** Grid of 10x10, and responsible for painting the cells depending on their types. Also, it contains methods related with mouse actions.
- **`class Algorithm –`** Contains two methods: *Dijkstra* and *AStar*. Depending on selection of the algorithm, it performs instructions and implements these algorithms.
- **`class Node –`** Attributes of cells, (e.g. their positions), calculations (i.e. distance) etc.

b. Type of the Cells

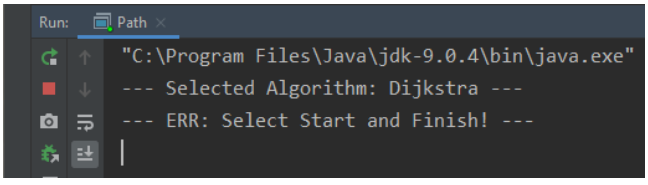
A cell can have the following types.



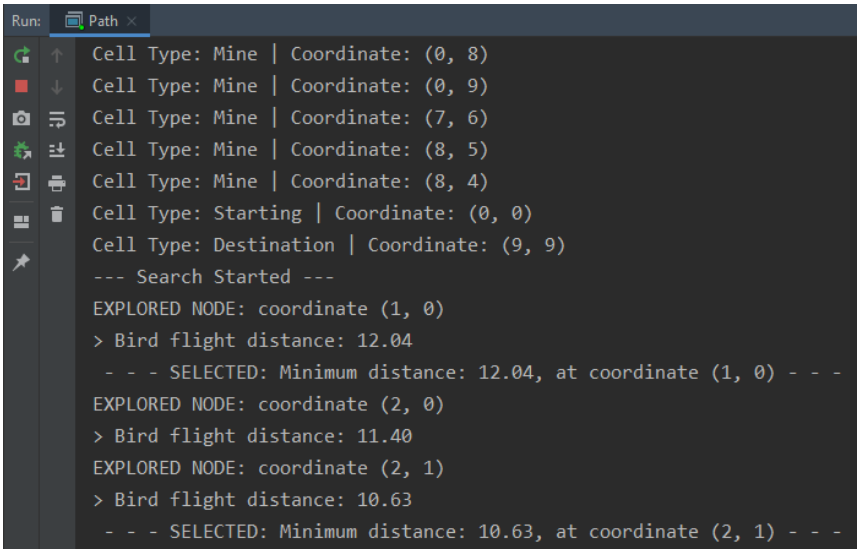
Empty Start Finish Mine Checked Found

c. Console

In order to provide detailed information regarding the program, we use console outputs. This is helpful when the trace of the program is desired, as well as debugging purpose. It also provides necessary instructions to the user such as, what is missing when he or she attempts to start the selected search.

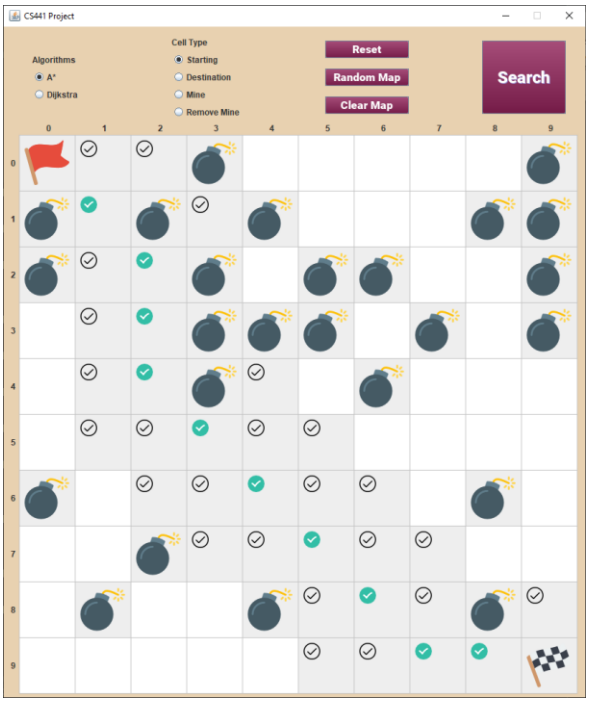


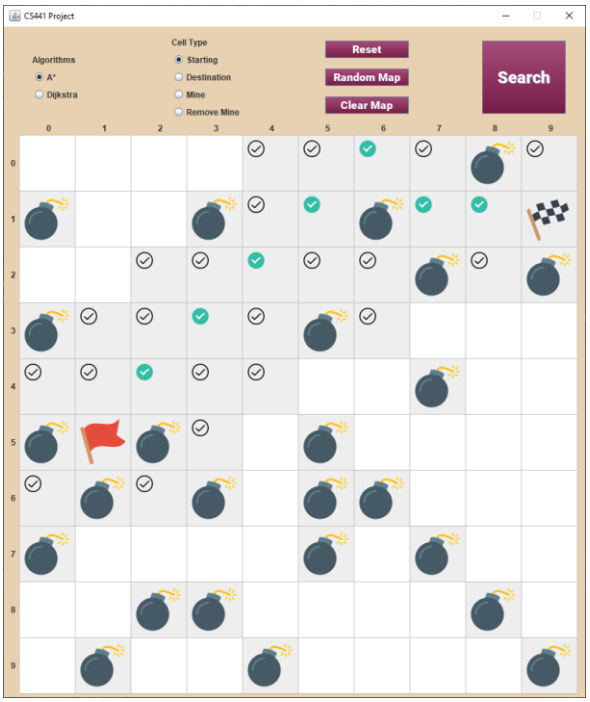
[Console]: Sample output (algorithm selection and error)



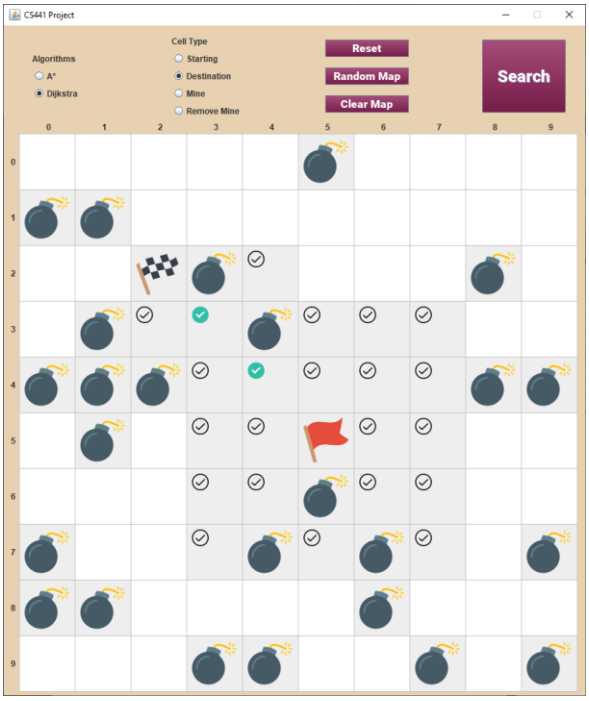
[Console]: Sample output (display of calculations, start of search, etc.)

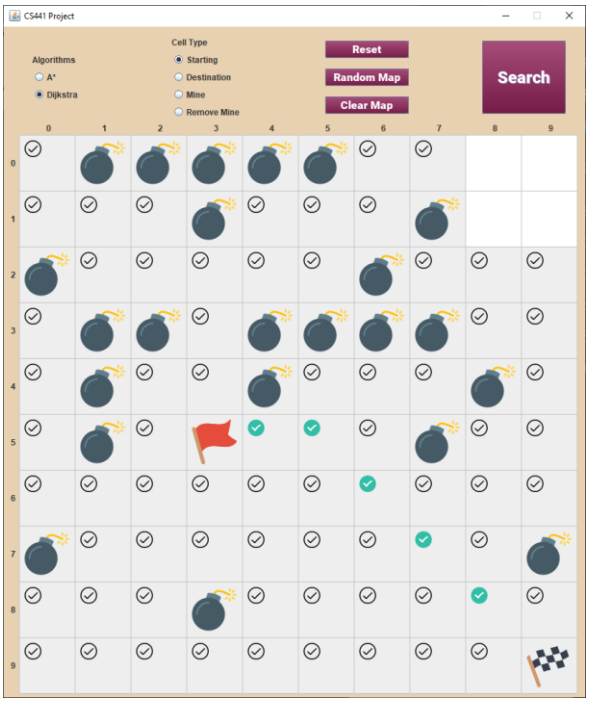
4. Testing





[Test]: A* Search Algorithms works flawlessly





[Test]: Dijkstra's Search Algorithms works flawlessly

5. References

[1] Goyal A, et all. 2014. *Path Finding: A* or Dijkstra's*

[2] Cormen T, et all. 2009. *Introduction to Algorithms 3rd Edition*

[3] https://www.101computing.net/a-star-search-algorithm

[4] https://docs.oracle.com/en/java

[5] http://en.0wikipedia.org

[6] Yapay Zeka 5.5 : A Yıldız (A*) Sezgisel Arama Algoritması, BilgisayarKavramlari

[7] Dijkstra en kısa yol algoritması Shortest path, BilgisayarKavramlari