

Modeling Dijkstra's Algorithm using Cell-DEVS CD++

AmirHoseein Ghorab

Nov 2020

Table of Contents

<i>Introduction</i>	<i>3</i>
Dijkstra's algorithm.....	3
Mapping the graph to 2D grid	3
Dijkstra's algorithm for 2D grids.....	4
<i>Formal specification of the coupled model.....</i>	<i>11</i>
<i>Extending Dijkstra's algorithm to use in maze (adding walls)</i>	<i>12</i>
<i>Creating maze using MazeMaker GUI.....</i>	<i>13</i>
<i>Known issues</i>	<i>16</i>
<i>Running and testing the Dijkstra CD++ model and YouTube Videos.....</i>	<i>16</i>
<i>Bibliography.....</i>	<i>17</i>

Introduction

Routing protocols have become an essential component in today's networks. These protocols are used to select paths for traffic between network nodes by using routing algorithms. Since the computer network can be modeled by a graph, the graph theory is widely used in routing algorithms. The routing algorithms are classified into two categories as follow:

- **Adaptive**
 - The routing decisions may select a new route for each packet in response to changes in the condition and topology of the networks.
- **Non-adaptive**
 - The routing decisions are not based on traffic, topology, or the current state of the network.

In this project, one of the non-adaptive routing algorithms called shortest path routing (Dijkstra's Algorithm) will be modeled by using Cell-DEVS.

Dijkstra's algorithm

This algorithm finds the shortest paths from a given source node to all other nodes (include destination node) starts from the source node and finds the nearest adjacent node. At the first iteration, the algorithm finds the nearest node from the source node which must be a neighbor of the source node. At the second iteration, the algorithm finds the second-closest node from the source node. This process will continue until all nodes are processed by the algorithm. During each iteration, the algorithm calculates the distance (cost) of each node from the source node. Later on, the algorithm uses these costs to find the shortest path to the destination [1, 2, 3, 4].

Mapping the graph to 2D grid

To model Dijkstra's Algorithm with Cell-DEVS, the graph needs to be mapped to the N-dimensional grid as shown in Fig. 1.

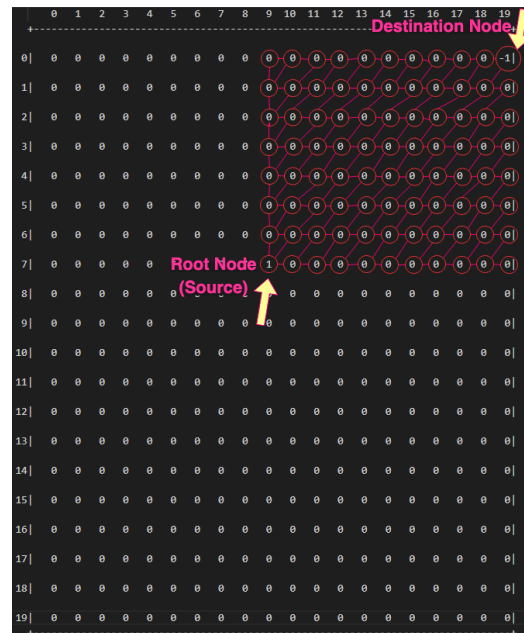


Figure 1: Mapping graph to grid

Dijkstra's algorithm for 2D grids

Cells' state assumptions

- **Source** cell's state(cost): **1**
- **Destination** cell's state(cost): **-1**
- **Dead** cell's (WALLs) state: **9999999**

Algorithm steps

1. Starting with the source cell (**cell with state 1**)

Figure 2: Before running the algorithm

0	0	0	0	0	0	-1
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

2. Increase the state of neighbors by **source cell's state + 1** during each iteration

Figure 3: Iteration #1

0	0	0	0	0	0	-1
0	0	0	0	0	0	0
0	0	0	2	0	0	0
0	0	2	1	2	0	0
0	0	0	2	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Figure 4: Iteration #2

0	0	0	0	0	0	-1
0	0	0	3	0	0	0
0	0	3	2	3	0	0
0	3	2	1	2	3	0
0	0	3	2	3	0	0
0	0	0	3	0	0	0
0	0	0	0	0	0	0

Figure 5: Iteration #3

0	0	0	4	0	0	-1
0	0	4	3	4	0	0
0	4	3	2	3	4	0
4	3	2	1	2	3	4
0	4	3	2	3	4	0
0	0	4	3	4	0	0
0	0	0	4	0	0	0

Figure 6: Iteration #4

0	0	5	4	5	0	-1
0	5	4	3	4	5	0
5	4	3	2	3	4	5
4	3	2	1	2	3	4
5	4	3	2	3	4	5
0	5	4	3	4	5	0
0	0	5	4	5	0	0

Figure 7: Iteration #5

0	6	5	4	5	6	-1
6	5	4	3	4	5	6
5	4	3	2	3	4	5
4	3	2	1	2	3	4
5	4	3	2	3	4	5
6	5	4	3	4	5	6
0	6	5	4	5	6	0

- After reaching the destination (cell with state **-1**), the algorithm moves backward from destination to source by passing through the cells with the minimum state (cost). **Notice that the algorithm finds all possible paths (if 2 or more cells have a similar state (cost) the algorithm passes through all of them)**. For example, in Fig. 7, the neighbors of the destination cell have state **6**, so in this case, the shortest path passes through both of them.

Moving backward:

I have developed a simple algorithm for moving backwards from destination to source shown in Fig. 8.

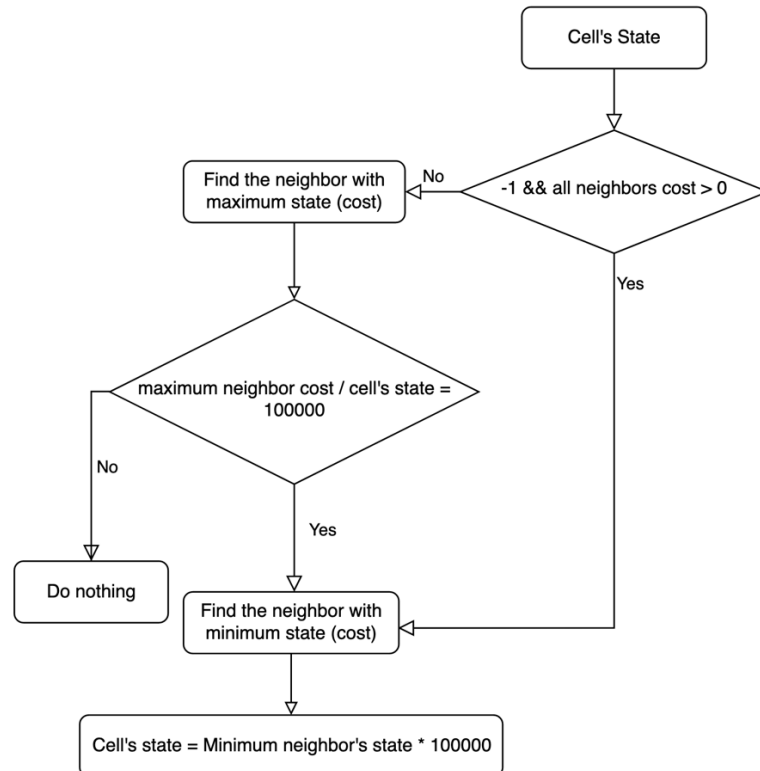


Figure 8: Moving backwards algorithm

For small grids the coefficient (100000) can be replaced to 10. Next iterations (moving backwards) after applying the aforementioned algorithm are shown in below figures.

Figure 9: Iteration #6

7	6	5	4	5	6	600000
6	5	4	3	4	5	6
5	4	3	2	3	4	5
4	3	2	1	2	3	4
5	4	3	2	3	4	5
6	5	4	3	4	5	6
7	6	5	4	5	6	7

Figure 10: Iteration #7

7	6	5	4	5	500000	600000
6	5	4	3	4	5	500000
5	4	3	2	3	4	5
4	3	2	1	2	3	4
5	4	3	2	3	4	5
6	5	4	3	4	5	6
7	6	5	4	5	6	7

Figure 11: Iteration #8

7	6	5	4	400000	500000	600000
6	5	4	3	4	400000	500000
5	4	3	2	3	4	400000
4	3	2	1	2	3	4
5	4	3	2	3	4	5
6	5	4	3	4	5	6
7	6	5	4	5	6	7

Figure 12: Iteration #9

7	6	5	300000	400000	500000	600000
6	5	4	3	300000	400000	500000
5	4	3	2	3	300000	400000
4	3	2	1	2	3	300000
5	4	3	2	3	4	5
6	5	4	3	4	5	6
7	6	5	4	5	6	7

Figure 13: Iteration #10

7	6	5	300000	400000	500000	600000
6	5	4	200000	300000	400000	500000
5	4	3	2	200000	300000	400000
4	3	2	1	2	200000	300000
5	4	3	2	3	4	5
6	5	4	3	4	5	6
7	6	5	4	5	6	7

Figure 12: Iteration #11

7	6	5	300000	400000	500000	600000
6	5	4	200000	300000	400000	500000
5	4	3	100000	200000	300000	400000
4	3	2	1	100000	200000	300000
5	4	3	2	3	4	5
6	5	4	3	4	5	6
7	6	5	4	5	6	7

As shown in Fig. 13, the algorithm finds all the possible shortest paths between source and the destination.

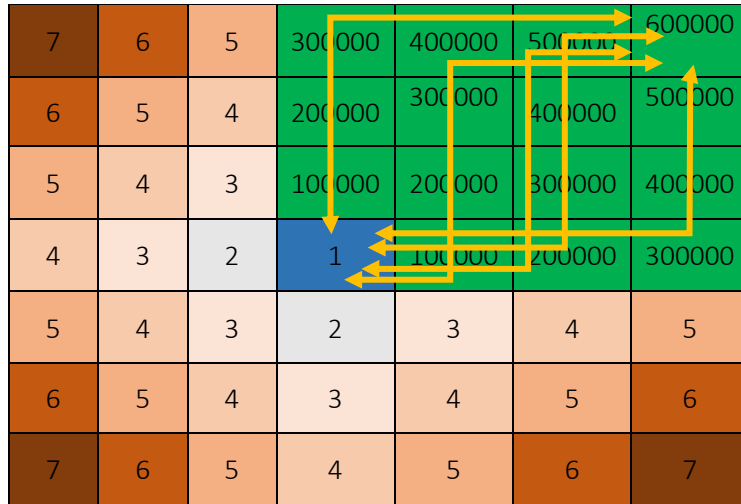


Figure 13: All the possible shortest paths

Formal specification of the coupled model

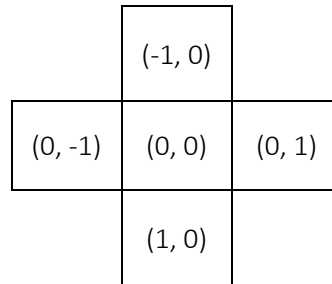


Figure 14: Neighborhood definition

$\langle X, Y, I, S, \theta, N, d, \delta_{\text{int}}, \delta_{\text{ext}}, \tau, \lambda, D \rangle$

$X = \emptyset$

$Y = \emptyset$

$N = \{(-1,0), (0, -1), (0, 0), (0, 1), (1, 0)\} \Leftrightarrow B = \text{No-Wrapped}$

$d = 100 \text{ Ms}$

$\tau: N \rightarrow S$

$S = \{$

- Source: 1
- Destination: -1
- Walls: 9999999
- Others: 2 to 9999998

$\}$

$R (\text{row}) = 10$ (can be changed)

$C (\text{column}) = 10$ (can be changed)

Extending Dijkstra's algorithm to use in maze (adding walls)

In order to extend the algorithm to use for pathfinding in the maze (adding wall), new rules have been added to the code.

rule : { (-1,0) + 1 } 100 { (0,0) < (-1,0) and (0,0) = 0 and (-1,0) != 9999999 }

rule : { (0,1) + 1 } 100 { (0,0) < (0,1) and (0,0) = 0 and (0,1) != 9999999 }

rule : { (0,-1) + 1 } 100 { (0,0) < (0,-1) and (0,0) = 0 and (0,-1) != 9999999 }

rule : { (1,0) + 1 } 100 { (0,0) < (1,0) and (0,0) = 0 and (1,0) != 9999999 }

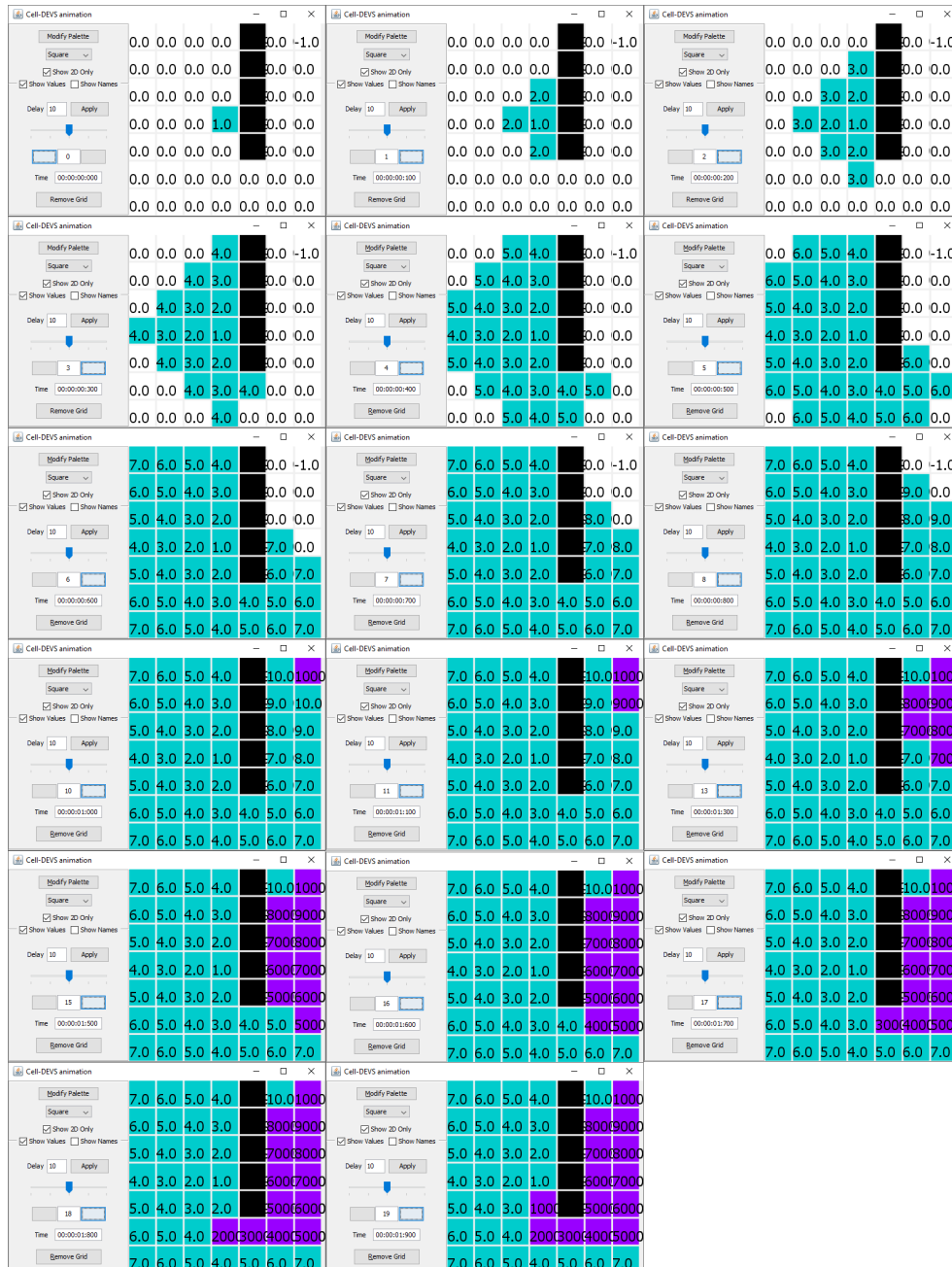


Figure 15: Adding walls (black squares)

Creating maze using MazeMaker GUI

It is always challenging to create different test scenarios for the model. To tackle this problem, I made a simple WebGUI application called MazeMaker to create different test mazes for the Dijkstra CD++ model. To access MazeMake open the below URL in a browser (guide videos are available in section **Running and testing the Dijkstra CD++ model and YouTube Videos**)

<https://nephilimboy.github.io/MazeMaker/>

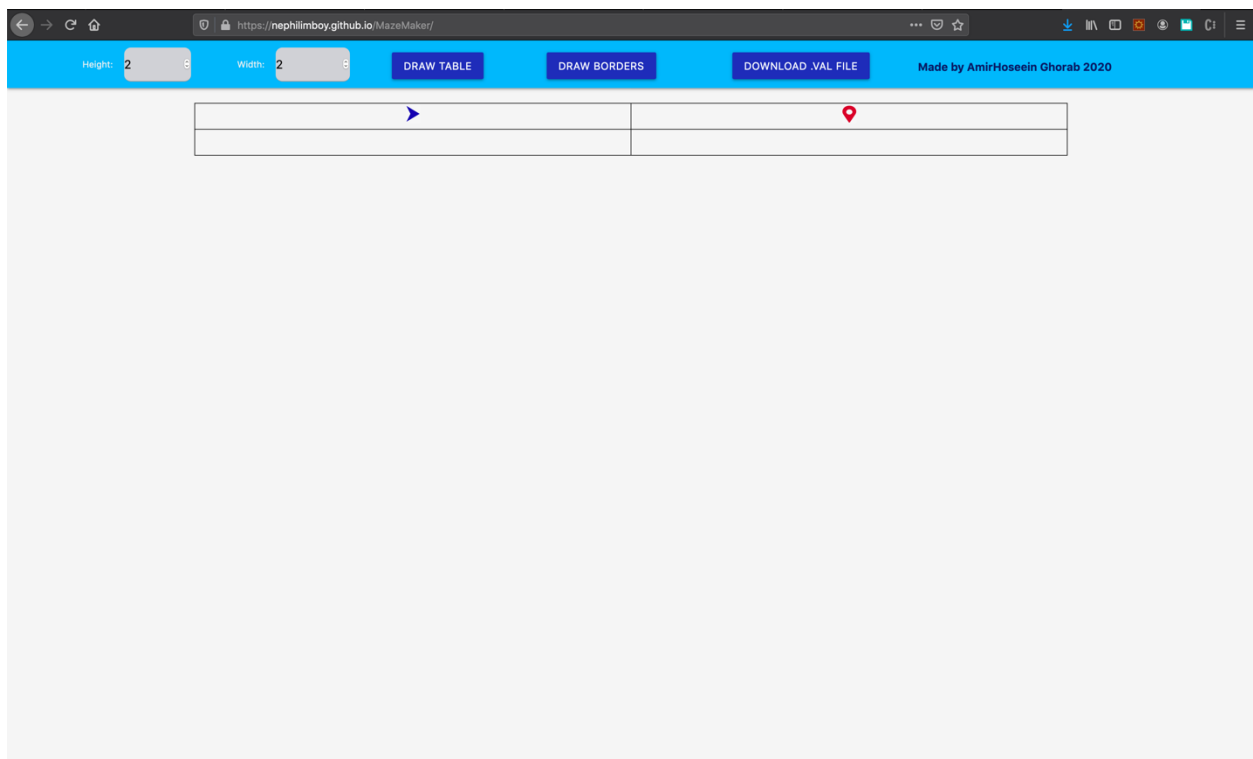




Figure 16: MazeMaker web GUI

- Height/ Width input box: Change size of the maze
- Draw Table button: Draw a maze based on height/ width input boxes
- Draw borders button: Fill the table's (grid) borders with walls (cells with state 9999999)
- Download .val file button: Download .val file to use for cell-devs model
- Source Cell: 
- Destination Cell: 

Drawing maze

- Source and destination can be moved by mouse drag and drop
- Walls can be added/deleted by clicking on an empty cell or moving the mouse pointer on the empty cells while holding down the right mouse button
- Walls can be added to the borders by clicking on the **Draw borders** button

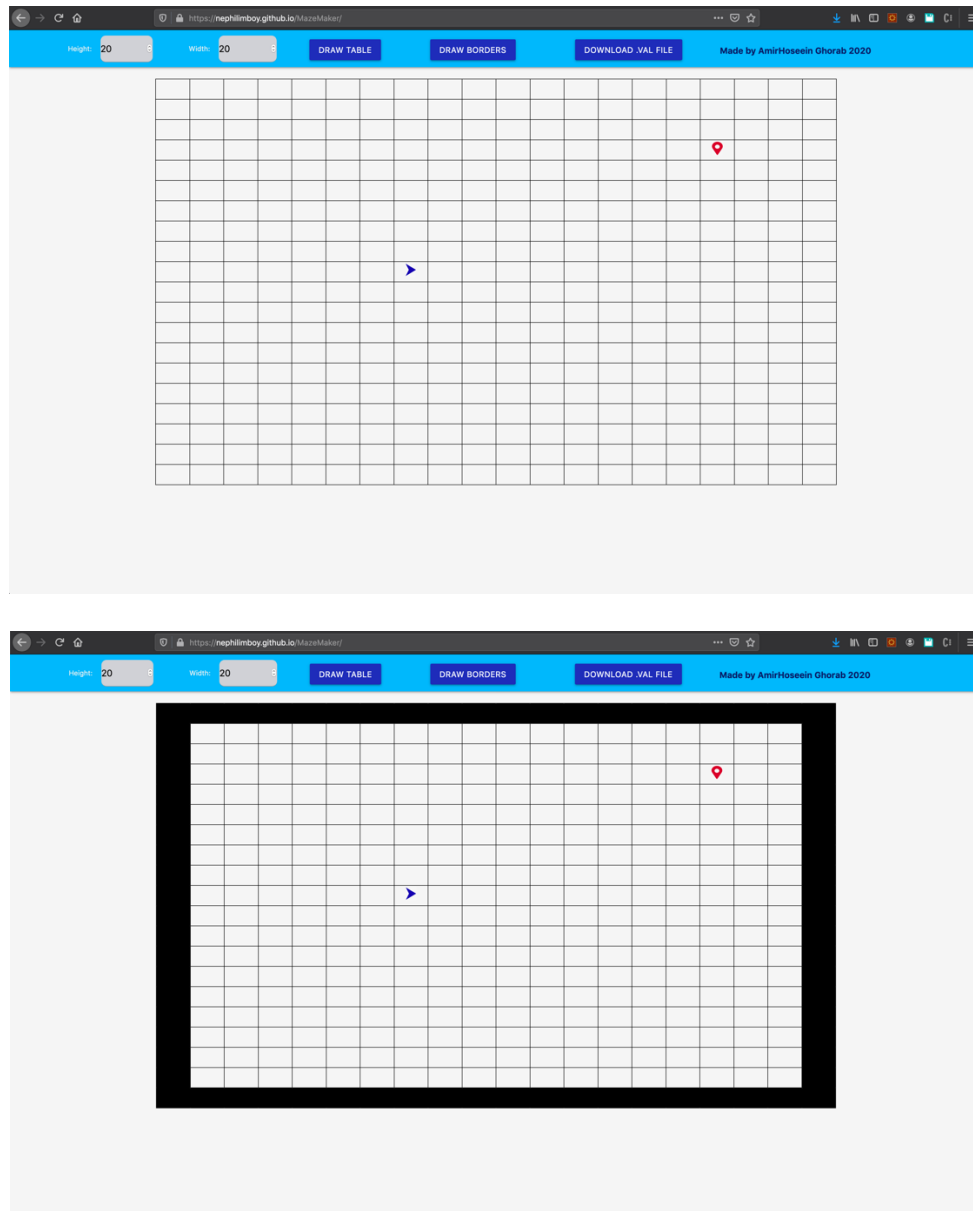


Figure 17: Adding the walls to the borders

Downloading .val file

The .val file can be downloaded by clicking on the **Download .val file** button.

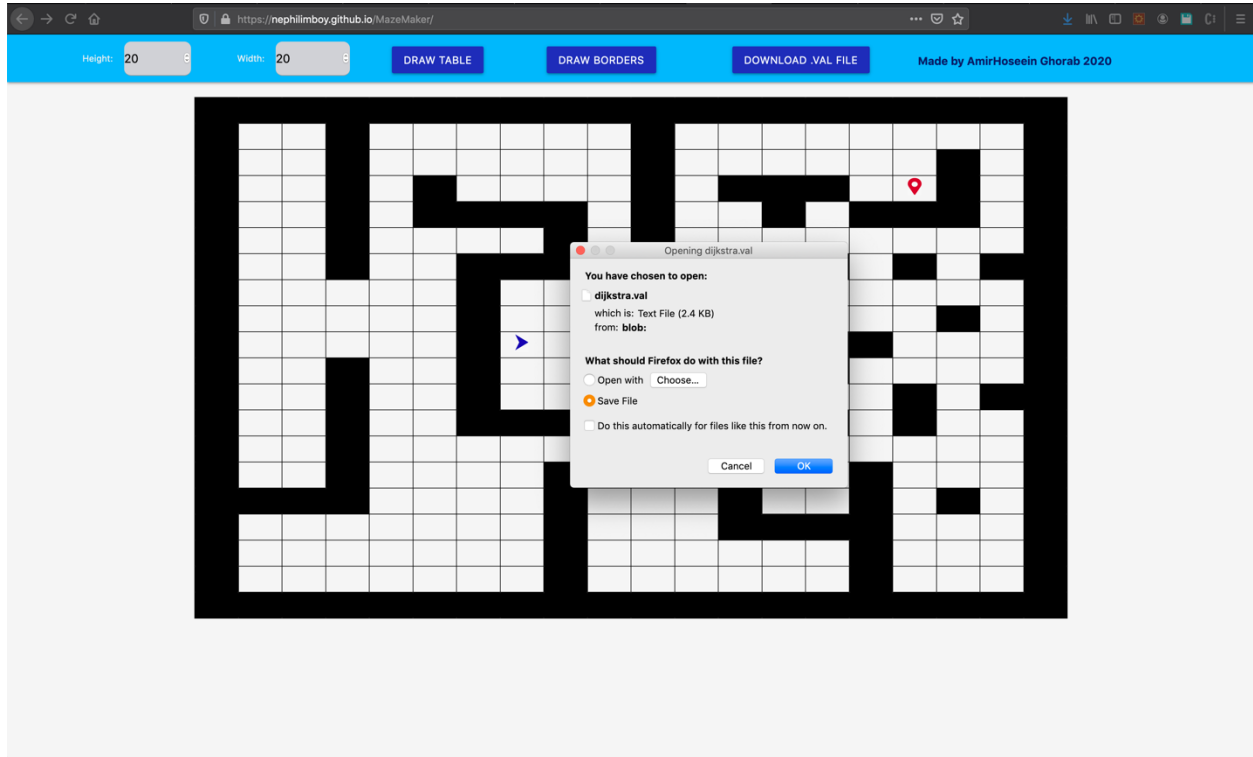


Figure 18: Downloading .val file

This application is hosted by GitHub and source code can be found <https://github.com/nephilimboy/MazeMaker>

Known issues

- If you want to use the extended Dijkstra CD++ model (adding walls to the model), make sure to fill the borders with walls to prevent unexpected bugs.
- Visualize the simulation with **WebViewer GUI** or **CD++ modeler (most recommended)** **not with GrafLog** (I don't know the exact reason why GrafLog crashes for me, but I guess it is because of the large integer number (**coefficient 100000**) that this model is using)

Running and testing the Dijkstra CD++ model and YouTube Videos

I have prepared multiple tests in the “**Tests/**” folder with the palate file for each. Also, there are some guide videos that I have uploaded on YouTube.

- <https://youtu.be/nZAGyJ9xwyk>
- <https://youtu.be/Aux7o-MXFU8>

Bibliography

1. Adamatzky, AI. 1996. "Computation of shortest path in cellular automata." *Mathematical and Computer Modelling* 105--113.
2. gettysburg. n.d. *cs.gettysburg.edu*.
<http://cs.gettysburg.edu/~jfink/courses/cs322slides/3-19.pdf>.
3. Wikipedia. n.d. *Wikipedia*. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm.
4. wordpress. n.d. *thiloshon.wordpress.com*.
<https://thiloshon.wordpress.com/2017/04/03/dijkstras-algorithm-for-path-finding-problems/>.