PATH FINDING ALGORITHMS VISUALIZATION

Artjom Valdas and **Kirill Grjaznov**

Institute of Computer Science, University of Tartu

UNIVERSITY OF TARTU 1632

Abstract

Project goal is to build the web application that helps visualize how path finding algorithms, such as Dijkstra and A* search, work in the maze environment. Also, allow users to place and remove barriers on their own. Finally, we want to compare these two algorithms efficiency.

Project idea

On the algorithmics course in one of the homework, we had to independently implement search algorithms. Like everyone else, of course, we got into the Internet to study and understand how these algorithms work. We randomly stumbled upon a web tool where we could choose different algorithms and run them on different playing fields, where we could set up barriers by ourselves. It helped us a lot to understand how these algorithms actually work and famous quote says *"better to see something once than to hear about it a thousand times"*.

When we had to choose a project, we decided that we wanted to make a similar cool tool, since we had never done anything similar before. We thought that it would be interesting for us to try to implement these algorithms in such an environment, as well as try to add an algorithm for creating a maze. It was a challenge for us and we also thought that it would be useful for future students who will take this course or just study this topic. Therefore, we decided to make this application on the web so that every person who has a computer, regardless of operation system, can use this tool.

Used algorithms

We decided to implement two algorithms for finding the shortest path in maze: Dijkstra and A* search. These two algorithms are good for comparison because the A* search is the evolved version of Dijkstra algorithm. For both algorithms nodes of a graph are the cells. Only orthogonally connected cells have edge so diagonal moving is excluded. All edges have weight 1 (distance between cells). Dijkstra algorithm are implemented without any modifications. In A* search algorithm each node has function $f = g + h$, were g is a distance traveled to reach this node and h is a Manhattan distance from this node to the finish node. We multiplied h by 1.1 to increase the cost for moving back from right direction. It shows more better results. For maze generating we used Recursive Division algorithm [1] because it makes possible to find path from any maze region to all other regions. So it is always possible to find path from start to end. Also Recursive Division algorithm generates more beautiful mazes because its fractal nature.

Web application

Our web application [2] is written from scratch without using any external libraries – only HTML (HyperText Markup Language), CSS (Cascading Style Sheets) and native JavaScript. The only thing that was added from the outside is Bootstrap 5 (CSS framework) for creating beautiful buttons in a user interface.

At the very beginning, we created an environment in which it was possible to draw any shapes and which could interact with the user. This allows the user to draw walls or obstacles that the algorithms will need to bypass. Also at the beginning, we placed two points - green and red, start and finish, respectively, as shown in *Figure 1*. In addition, the user can move these two points wherever he/she pleases, just by clicking on and moving them. If the user does not want to draw walls, he can take advantage of the possibility of creating a maze using our implemented algorithm. The beauty is that the user can also specify the difficulty of the maze - the harder the level, the more intricate the maze.

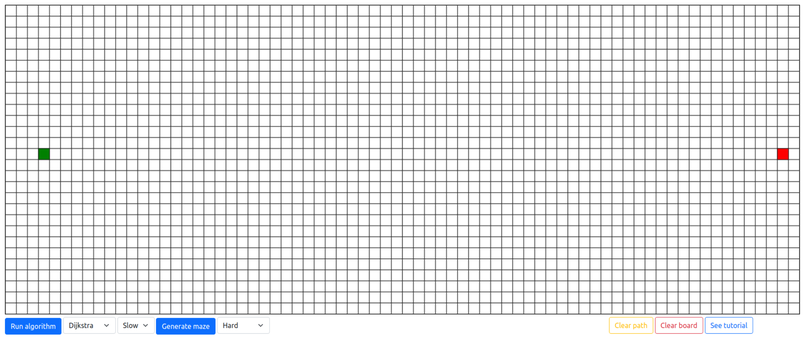

Fig. 1: Web application.

In addition to creating walls, just by pressing the mouse button down and dragging it around the playground, the user also has the option to erase them. To do this, he needs to click on any wall and start moving the mouse. So you can erase down any amount of cells. If you want to erase all the walls at once, you can click on the "*Clear board*" button and all the walls will disappear.

Now speaking about one of the most important parts of this project - the ability to run and visualize search algorithms. As previously mentioned, the user has a choice between two algorithms. To do this, the user simply selects the algorithm of interest and presses the "*Run algorithm*" button. The cells that the algorithm has visited will be repainted into light-blue, and the shortest path found will be shown in yellow squares. Another cool opportunity for the user is the ability to control the rendering speed. There are only two of them, but this is enough to either understand how the methods work or to see how they solve tasks.

And finally, the application also has a button "*See tutorial*", which opens modal. A short explanation of how to use this application is written in the open window. This should be enough to understand how everything works and how to use this application.

Comparison

In the pictures below you can see how much nodes are visited by Dijkstra (first picture) and A* search (second picture) algorithms. You can see that Dijkstra algorithm visits more cells. This is due to the fact that this algorithm does not know in which direction it should move.

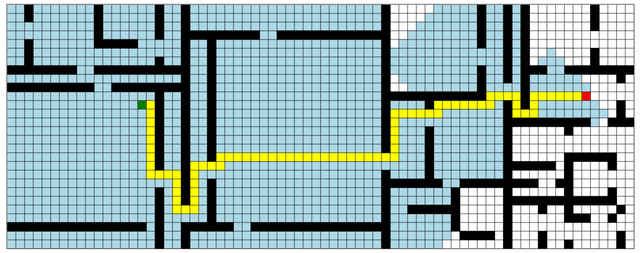

Fig. 2: Dijkstra algorithm.

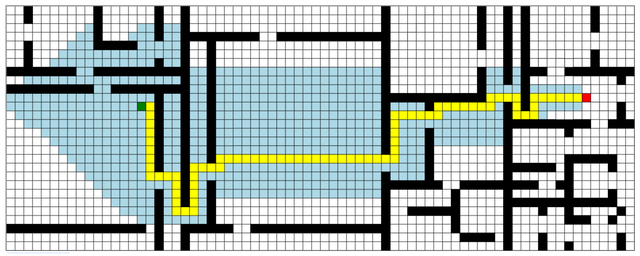

Fig. 3: A* search algorithm.

Results

As result, there is ready to use web application for visualizing the Dijkstra and A* search algorithms, that can help new students understand these path finding methods. Clearly visible, both algorithms always find the same path, but Dijkstra algorithm is less efficient, because it visits more cells.

References

[1] Jamis Buck. *Mazes for Programmers*. URL: https://weblog.jamisbuck.org/2011/1/12/maze-generation-recursive-division-algorithm.

[2] KilRil valart. *Pathfinding*. https://github.com/valart/pathfinding. 2022.