TDT4136 A* Exercise

Sigve Skaugvoll

October 6, 2016

Abstract

The goal of this assignment is to implement the A* algorithm, and use the implementation to find the best path from a start node to a goal node. A* is a best first algorithm, which considers something called the gscore, fscore and heuristics, to calculate the best path.

A: Pathfinding in 2D games

A common demonstration problem for A* is that of finding shortest paths in two-dimensional square-grid boards. Find shortest paths in grids with obstacles. Using your A* implementation, search for a path from the start cell to the goal cell.

A.1.2 : Grids with Obstacles

I implemented the A* algorithm my self. I have commented my code good as I can. Explaining what each variable, object and function does. and how it does it.
For the implementation, and GUI I've used the programming language Java, JavaFX, and Intellij IDEA.

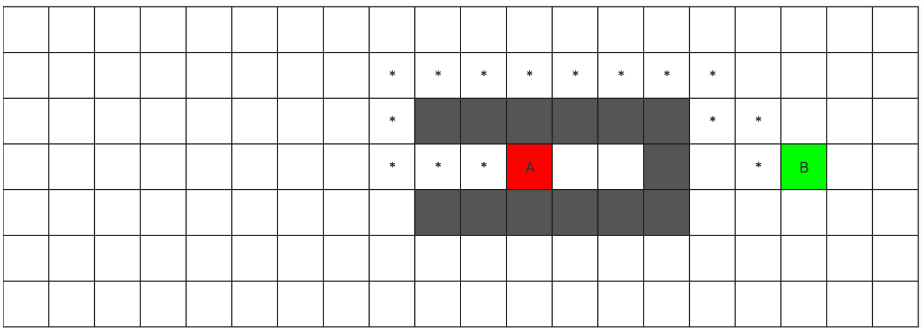

Figure 1: A.1.1 : Solved using A*

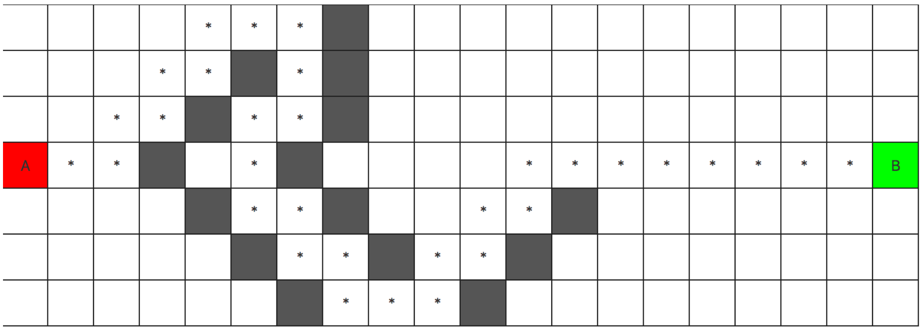

Figure 2: A.1.2 : Solved using A*

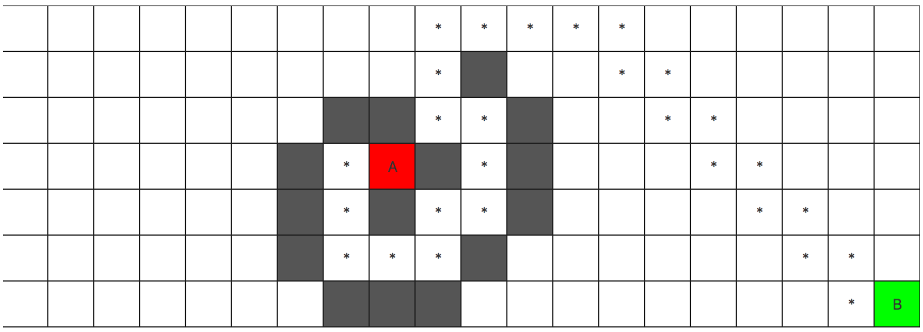

Figure 3: A.1.3 : Solved using A*

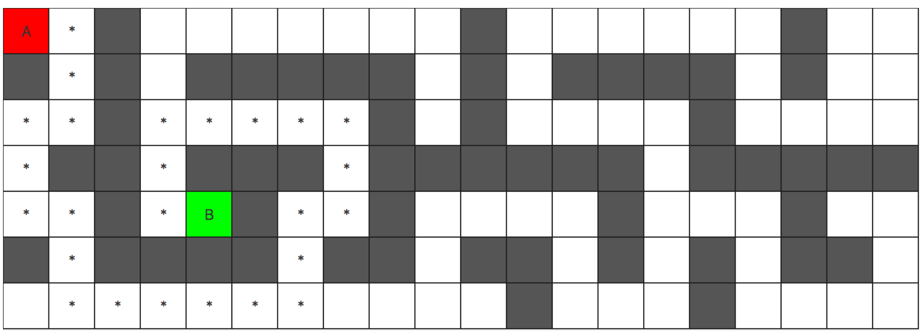

Figure 4: A.1.4 : Solved using A*

A.2 : Grids with Different Cell Costs

A game where the squares on the game board have different costs attached to them.

In this exercise the board represents an outdoor environment, different squares contain different kinds of landscape such as forest, mountains, grasslands, etc. Walking across a square of mountains would naturally take a longer time than a square of grasslands. Thus, a mountain square should have a higher cost attached to it than a grasslands square. Table 1 specifies the cell types that should be supported,

Table 1: Cell types and their associated costs.

Char.	Description	Cost
w	Water	100
m	Mountains	50
f	Forests	10
g	Grasslands	5
r	Roads	1

For this exercise I use the same A* implementation as in the previous mentioned. The results generated by the algorithm implementation is as follows:

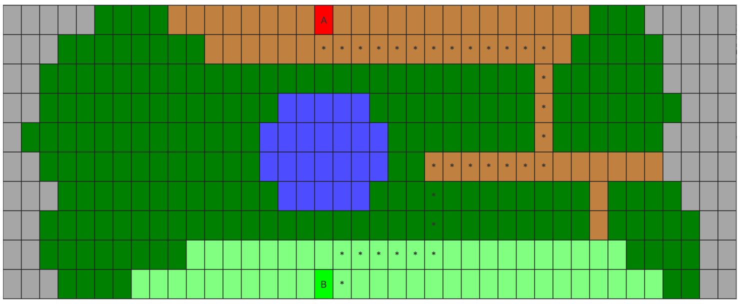

Figure 5: A.2.1 : Solved using A*

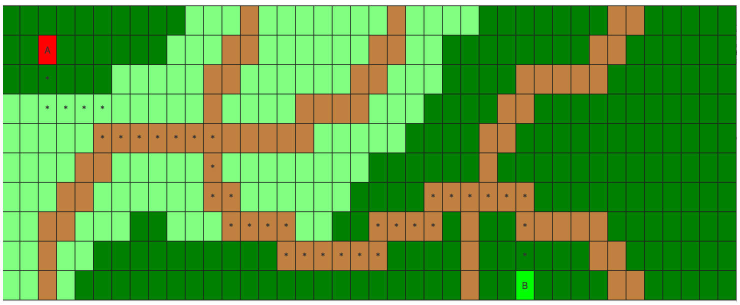

Figure 6: A.2.2 : Solved using A*

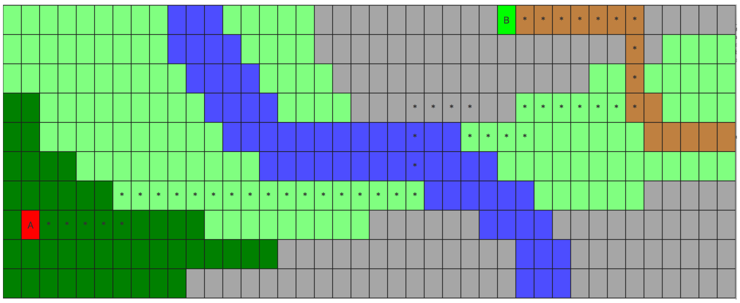

Figure 7: A.2.3 : Solved using A*

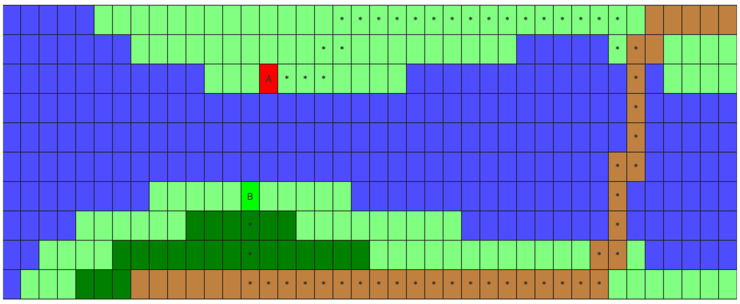

Figure 8: A.2.4 : Solved using A*

A.3 : Comparison with BFS and Dijkstra's Algorithm

The A* algorithm can, with a few changes, be modified to instead implement Breadth-First Search (BFS) or Dijkstra's Algorithm. With A*, the open nodes are sorted according to their expected cost f(s) = g(s)+h(s). By maintaining the list of open nodes as a queue (first-in first-out) instead of as a priority queue, the algorithm becomes BFS. By sorting the open nodes according to only g(s), the algorithm becomes Dijkstra's Algorithm.

To turn on the visibility of the open-nodes and the closed nodes. you have to change the variable "comparison" to "true" in the "DrawBoard" class. [/src/Astar/Controllers/DrawBoard.java, Line 19]

A.3.1 : Algorithms with open, closed and path visible

* = Open, x = Closed, black circle = path.

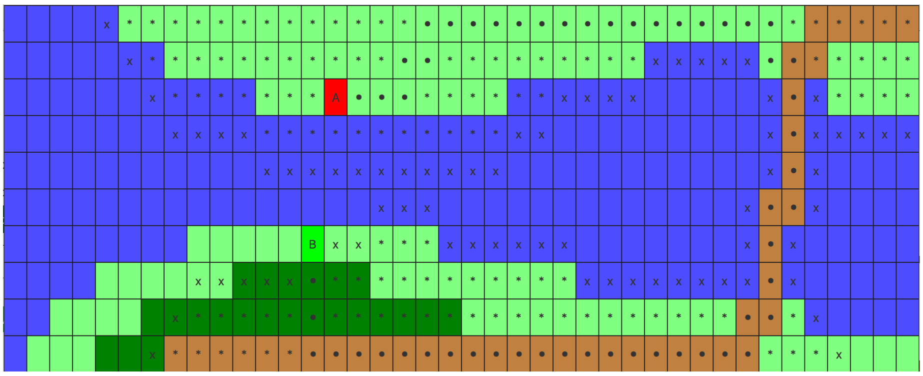

Figure 9: Board A.2.4, Algorithm: A*

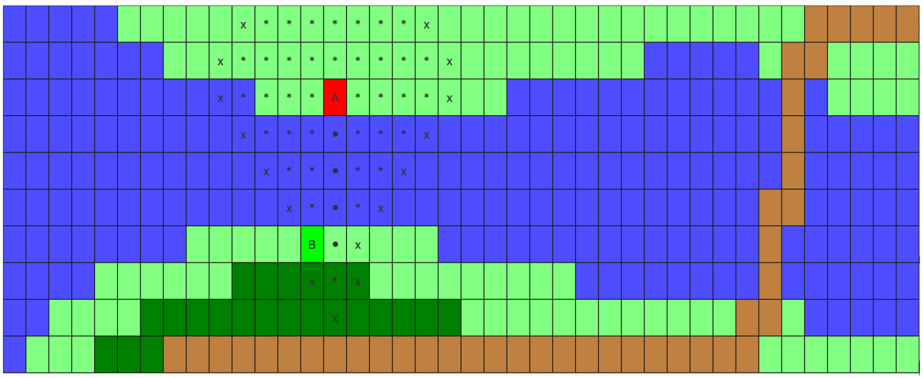

Figure 10: Board A.2.4, Algorithm: BFS

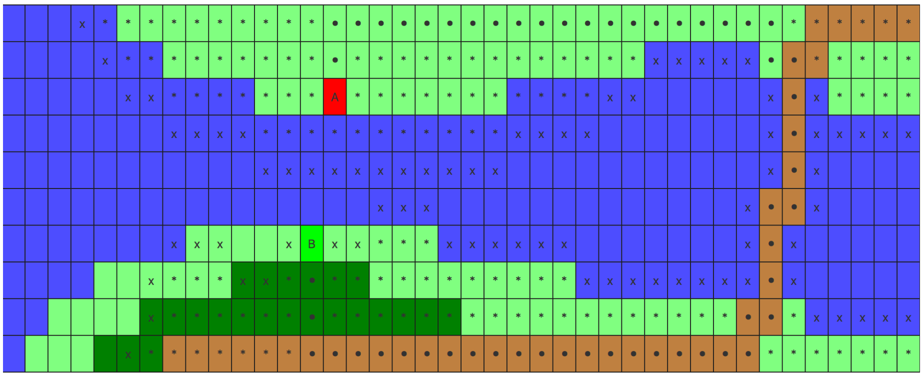

Figure 11: Board A.2.4, Algorithm: Dijkstra

A.3.2 : Visualization of each board using all tree algorithms

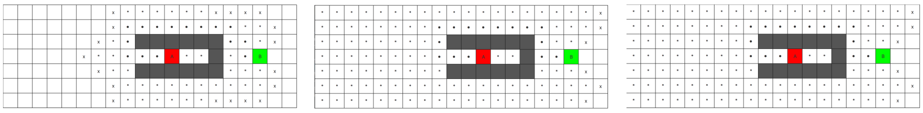

Figure 12: Board A.1.1, Left to right: A*, BFS, Dijkstra

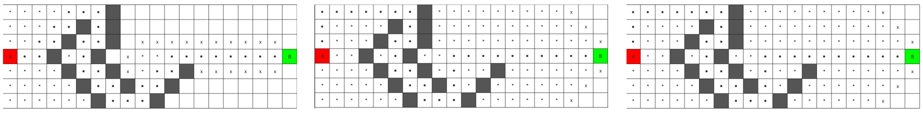

Figure 13: Board A.1.2, Left to right: A*, BFS, Dijkstra

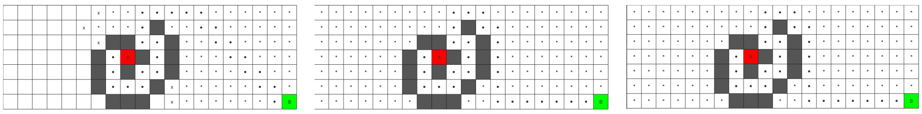

Figure 14: Board A.1.3, Left to right: A*, BFS, Dijkstra

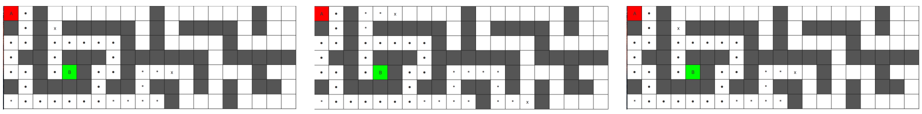

Figure 15: Board A.1.4, Left to right: A*, BFS, Dijkstra

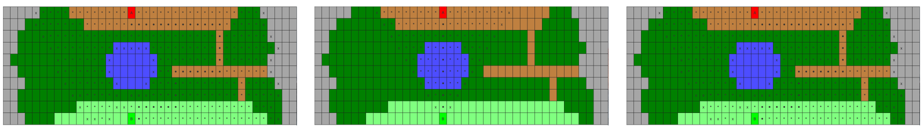

Figure 16: Board A.2.1, Left to right: A*, BFS, Dijkstra

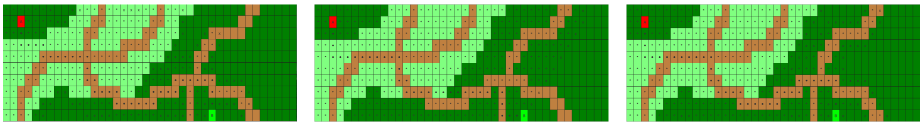

Figure 17: Board A.2.2, Left to right: A*, BFS, Dijkstra

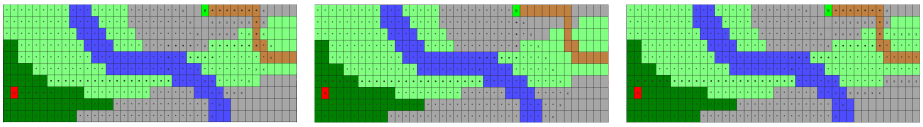

Figure 18: Board A.2.3, Left to right: A*, BFS, Dijkstra

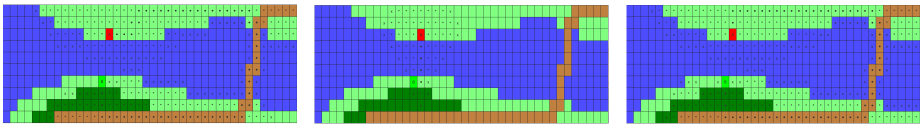

Figure 19: Board A.2.4, Left to right: A*, BFS, Dijkstra

A.3.3 : Difference between A*, BFS and Dijkstra

Board	***Diff in path***	***Diff in open***	***Diff in closed***
A.1.1	All algorithm uses same amount of steps from start to goal. A* takes a different path, than the one BFS and Dijk. takes	A* opens less neighbors than BFS and Dijk, Although BFS and Dijkstra opens same amount of neighbors	A* closes more neighbors than the other algorithms, BFS and Dijk. closes same amount.
A.1.2	All algorithm uses 32 steps. A* takes a route that gets closer or has same heuristic distance to goal. BFS and Dijkstra takes same route, but does not get closer to the goal, from start. Doesn't consider heuristic	A* differs from BFS and Dijkstra. Opens less neighbors, uses heuristics to calculate which neighbor to "open".	A* differs from BFS and Dijkstra. Closes less neighbors, goes for best route!
A.1.3	A* takes a route that get closer and closer to the goal, for almost every step in a heuristic distance mater, but BFS and Dijkstra goes for a more "straight line" path. and tries to go as straight as possible at any given time.	A* differs from BFS and Dijkstra that are the same. A* uses less neighbors	A* differs, it opens a few more neighbors.
A.1.4	No difference.	A* opens 8 neighbors, while BFS and Dijkstra opens 16. A* stops because it closes the neighbor when it no longer leads to a better path. BFS and Dijkstra opens new neighbors, until they find the goal.	No difference.

Table 1: Analysis of boards from A.1.x

Board	***Diff in path***	***Diff in open***	***Diff in closed***
A.2.1	A* and Dijkstra is equal in path cost. but takes a different route in "environment"-when the cost is the same, but uses same amount of steps. BFS doesn't consider weights, so it takes the shortes path, but a really expensive one!	A* and Dijkstra is equal, BFS opens less neighbors because it finds the goal path faster, and therefor doesn't have to open more neighbors to find goal.	BFS differ, in the way that it closes less neighbors.
A.2.2	A* and Dijkstra finds the same path. BFS takes a different route, because it doesn't consider the weights.	A* and Dijkstra doesn't have much difference when opening neighbors. A* opens fewer. BFS opens more neighbors, therefor visits more of the board. Dijkstra also visits little more of the board than A*	A* closes more than Dijkstra that closes more neighbors than BFS.
A.2.3	All paths are different. A* uses 2 'w', and 4 'm', has a subcost of 400. Dijkstra uses 4 'w', has a subcost of 400, the rest is equal, both find cheapest path. BFS finds shortest path in terms of steps.	A* and Dijkstra has minimal difference. The difference between A*, Dijkstra and BFS when opening neighbors. BFS opens less then Dijkstra, but more than A*.	BFS closes less neighbors than A* and Dijkstra. A* and Dijkstra closes the same neighbors.
A.2.4	No equal paths. The path cost of A* and Dijkstra is the same, so no route is better than the other, A* gives a feeling of closing on the goal, from start, then Dijkstra. BFS finds the best path in steps, but not necessarily cost.	A* opens the same neighbors, as Dijkstra, but Dijkstra adds additional three more neighbors. BFS opens remarkably less neighbors than the other algorithms. This is positive in terms of step cost and running time.	A* and Dijkstra closes almost the same neighbors, but A* closes a few less neighbors than Dijkstra. So A* has a few less steps than Dijkstra in finding the goal. BFS closes less neighbors than the others.

Table 2: Analysis of boards from A.2.x