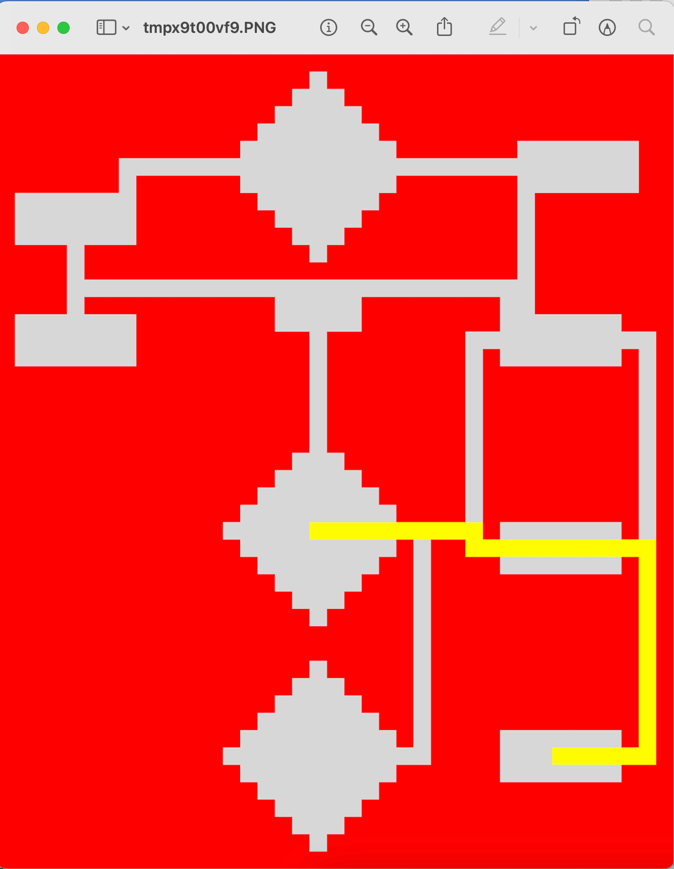
Introduction to AI

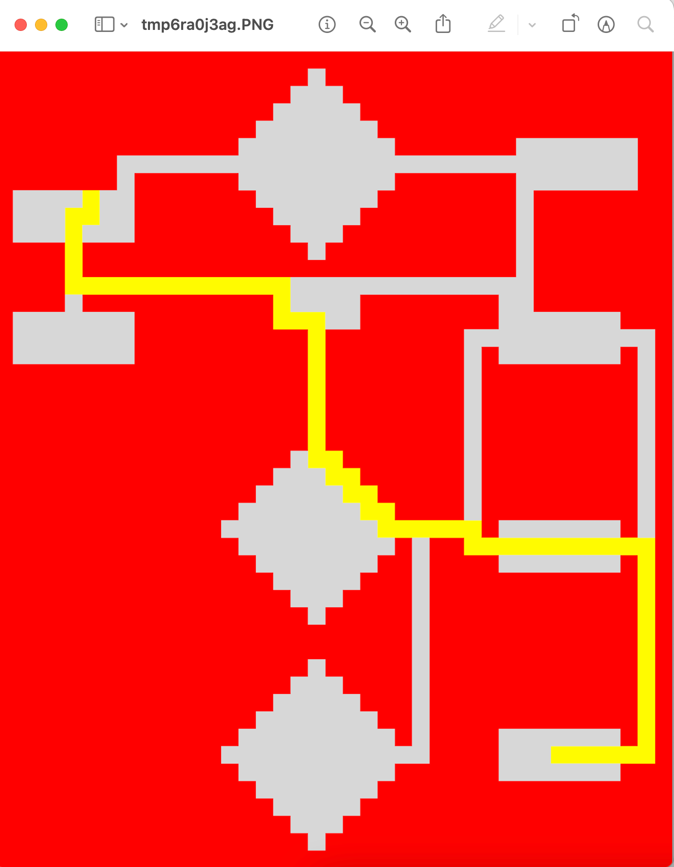
Assignment 2 by Jonas B Olsen and Emil Bjørlykke Berglund

We read this wiki article [https://en.wikipedia.org/wiki/A\*\_search\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)

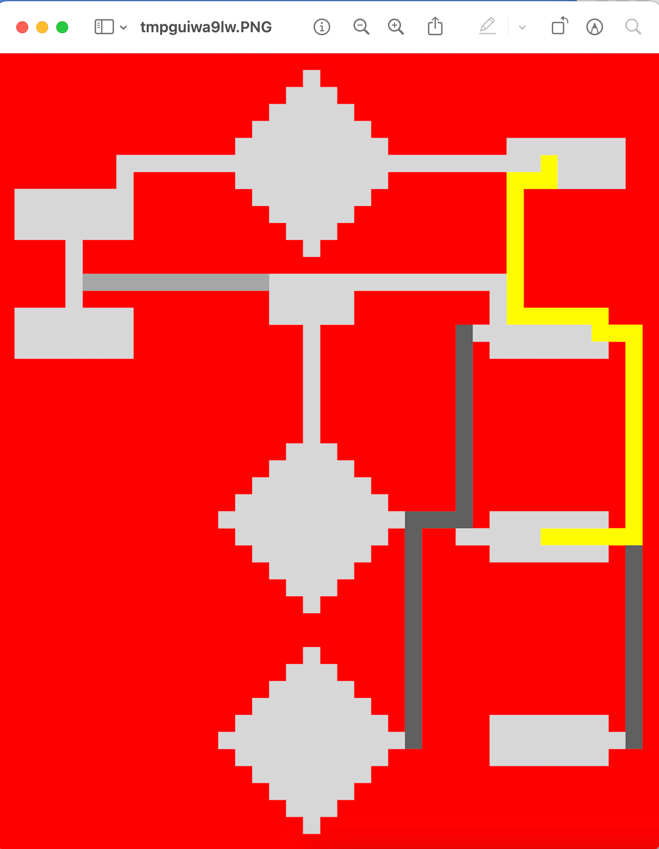
and looked at the AIMA chapter on A\* to understand what we were asked to implement.

**Task 1:**

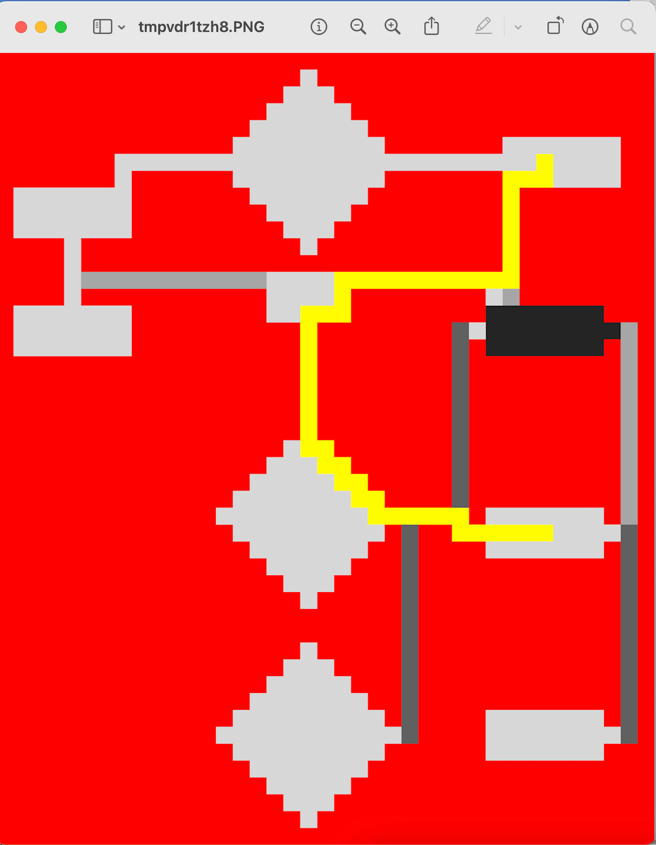
**Task 2:**

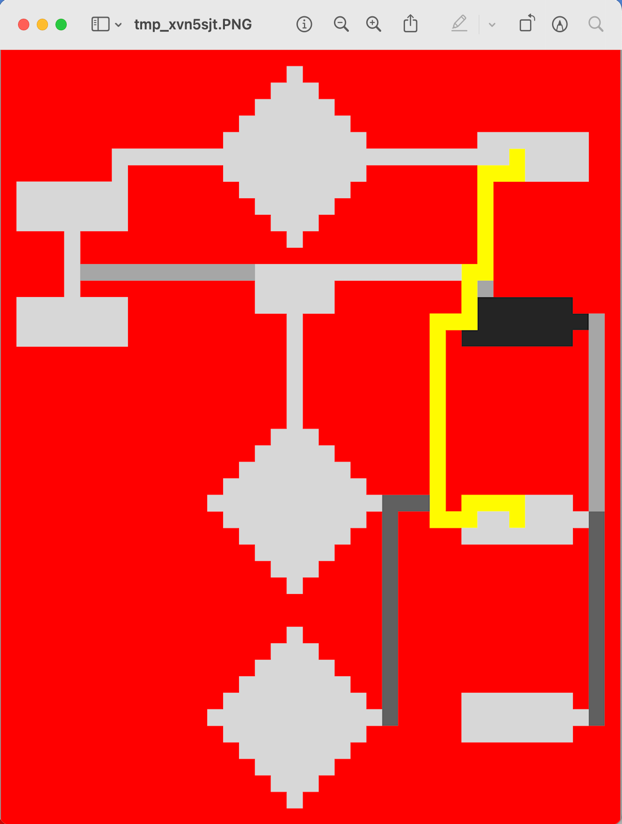


**Task 3:**



**Task 4:**





**Note on task 4:**

Adjusting the algorithm, we got it to take one of two ways. The left route had a final g-score of 64 whilst the right one had a final g-score of 70. We therefore concluded that the left route is the optimal one and kept this version of the algorithm. This version is also the one giving all the other paths.

We used the Eucledian distance to compute H-score

**About the code:**

To run you need to have the following python packages:

* Numpy
* Math
* Pillow
* Pandas

**Running the code:**

Et bilde som inneholder tekst

Automatisk generert beskrivelse

These two lines runs the code. Change the task = NUMBER to change between the tasks. You also need to go to the end of the string path in the .read\_map() and set the correct map. Everything is described with comments in the code as well.

task = 1 ⬄ .../Samfundet\_map\_1.csv

task = 2 ⬄ .../Samfundet\_map\_1.csv

task = 3 ⬄ .../Samfundet\_map\_2.csv

task = 4 ⬄ .../Samfundet\_map\_Edgar\_full.csv

**Description:**

The following logic is what we (at least tried) to implement:

We have 3 lists keeping track of the nodes, consideration, visited and path. (Though we really don’t need the last one)

When choosing a new square, we add this to the visited list, and add neighboring squares that are not already in visited or consideration, also needs to not be a wall, into consideration list. Then we look through consideration and picks the path with lowest f-cost. We also needed to implement a check if the previous and current square we chose is the same. This was to avoid getting stuck in corners. If this happened, the code adds this point to visited, removes this square from consideration, and search through the consideration list again to search for the then next-best square.

Nodes are given their g-, h- and f-cost upon initialization.

When a path is found, the code runs backwards, looking at each node´s parent-node, and adds these to the path-list. All the nodes in path sets the value in the grid to 5, giving the yellow path.

The drawPath method we “borrowed” from the given Map.py ☺