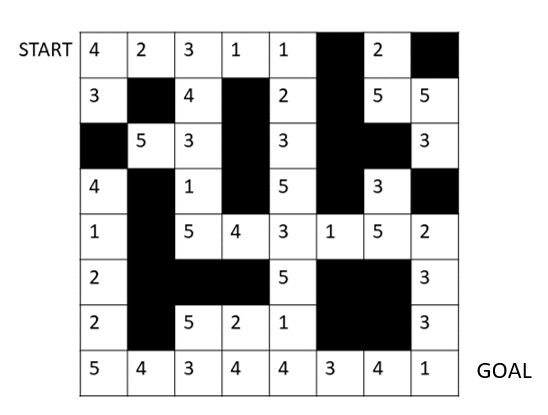
# INFO 180 – Methods in Artificial Intelligence

## Assignment 2: Search algorithms – Obligatory assignment

In this assignment, you will get to try to program a bit a simple artificial intelligence application and test out search algorithms.

The program you will work with is a program that finds solutions for a search in a grid of obstacles. One can imagine a robot that has to move in a grid of size N x N, where many of the fields in the grid are not accessible. In addition, there is a cost to moving into a field. You see an 8x8 example in the figure below.



The goal is therefore to find a path through the grid from the starting position at the top left (coordinates x=0, y = 0) and down to the target position at the bottom right (coordinates x=N-1, y = N-1). In each field in the grid, the robot can move in all directions (right, left, up, down, diagonally), i.e. 8 possibilities.

In the file week36.zip you find some files that you will use to check how well different search algorithms work on different sizes etc. on the grid. These programs are not complete and you must complete them. The goal is to achieve a comparison of breadth-first, depth-first, lowest-cost-first, and variants of searches with different heuristics.

**grid\_walk\_space.py** contains a class that formulates the problem in a search space and finds a solution with the solve method. solve takes two arguments (optional)

*frontier* which is an object of the type of data structure that realizes breadth-first (in python Queue()), depth-first (in python is LifoQueue(), i.e. a stack), and lowest-cost-first (in python PriorityQueue())

*heuristic* which is an object of type Heuristic. Heuristic contains only one static method that is used together with priority queue to implement heuristic search.

**grid\_walk\_node.py** contains data about a node in the search and functions that make it possible to create neighboring nodes in the grid.

**grid\_walk\_path.py** contains data about a path in the search, i.e. nodes in the path, cost for the path, and heuristic value for the path.

**make\_grid.py** contains data and functions that create a random grid that is used in the search. In the function make\_random\_grid(), the size of the grid is set, the probability that a field is unavailable, and a so-called seed for initializing the random number generator. You can vary this in the program line where it says

THE\_GRID = make\_random\_grid()

by giving values to the three optional parameters size, prob and seed.

**heuristic.py** defines a class Heuristic. It is very simple and only has one function that defines a heuristic for the search.

**zero\_h.py** contains a subclass of Heuristic that defines a heuristic that is 0 for all nodes. It thus behaves identically to Heuristic()

Task 1:

In **grid\_walk\_node.py**, 5 methods for moving in the grid are not finished. These methods have a 'return None' line. Finish them.

Task 2:

In **grid\_walk\_space.py** there is a "main" method at the end of the file which runs a breadth-first search and prints time spent in seconds, nodes visited, length of path and cost of path. It should work if you have managed to solve task 1 satisfactorily, but it can quickly take some time. Add program lines that also run the search with Depth-first search and Lowest-cost-first search. Compare the search methods based on time, path length and path cost.

Task 3:

Create a new heuristic in the form of the class *ZeroH* in **zero\_h.py**. This heuristic is to be used in a heuristic search. The heuristic must calculate the difference between the x-coordinates and y-coordinates of each position (node) and the target position, and then use the larger of these two values. Test the heuristic as in task 2 with the program line

*sp = GridWalkSpace(PriorityQueue(), YourHeuristic())*

Task 4:

Also create a new heuristic that calculates what we call the Euclidean distance between each position and the target position. This means that you must use Pythagoras. Also test this heuristic.

Task 5:

Create a new heuristic that does the same as the one in Task 3, but multiply by 3. Test this heuristic.

Task 6:

Vary size, probability, seed in file **make\_grid.py**. Which of the heuristics you have programmed do you think works best? Are any of the heuristics admissible so that you have an A\* algorithm.