

# Lab Assignment 2

February 15, 2010

In this lab assignment, we are going to use the A\* search algorithm to plan the shortest path from a start to goal position in a two-dimensional environment. The input environment is specified by a text file, and the output is a copy of this environment to which the planned path is added (if one exists).

## The Map

Our two-dimensional environment is composed of two types of tiles, namely walls and floor, and can be described by a text file. The left of figure 1 shows an example of such a text file. Each '\*' character represents a wall tile, and each white space (' ' character) represents a floor tile. The job of our path planner is to find a path of adjacent floor tiles that connect start tile with the goal tile. These two tiles are specified in the text file by the characters 'S' and 'G'.

The tiles effectively describe a rectangular grid, as shown in the right of figure 1, where each grid point corresponds to either a wall (red dots) or floor tile (green points). Consequently, a path connecting 'S' with 'G' is formed by a sequence of adjacent floor grid points. You can use this example, named 'floorplan.dat', which can be downloaded from the course website.

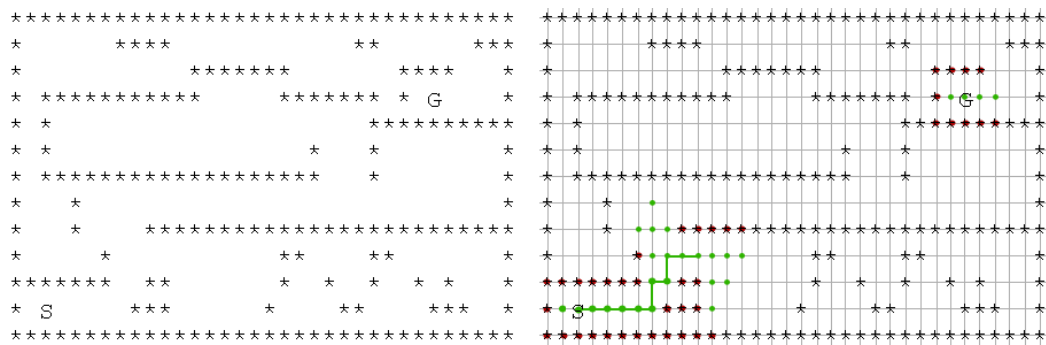


Figure 1: The left figure shows an example input. The right figure illustrates the grid over which we can move. Two portions of the map are colorized to illustrate wall and floor grid points. The green (partial) path is formed by adjacent floor grid points.

The first part of the assignment is to read the environment from 'floorplan.dat'. A good structure to store the imported information is a two-dimensional `char`-array; the array indices then effectively specify the  $x$  and  $y$  coordinates of a environment tile. You can assume that the map is rectangular and that its height and width are specified on the first and second line of the text file, respectively. E.g., the map of 'floorplan.dat' consists of 13 lines of each 34 characters.

*Tips:*

- Note that each line of ‘floorplan.dat’ actually contains 35 characters: the first 34 are “tile”-characters and the last character is a newline character (which, depending on your platform, can be ‘\n’, ‘\r\n’ or ‘\r’)
- The `get` method from the `ifstream` class reads one character.
- Function `atoi` converts a string to an integer.

### A\* Search Algorithm

After identifying the coordinates of the start and goal, the second part of the assignment is the implementation the A\* search algorithm. You will find an abundance of information on this algorithm online. Good sources are the following:

A detailed explanation: <http://www.policyalmanac.org/games/aStarTutorial.htm>

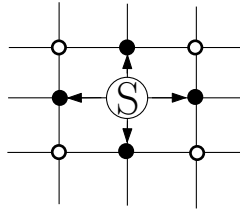
Algorithmic outline: [http://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](http://en.wikipedia.org/wiki/A*_search_algorithm)

Our path planner starts at the start node (tile) and can “walk” over the grid to adjacent floor tiles; i.e., from a given node, the planner can move in four directions: east, west, north and south (left of Fig. 2). Consequently, you can consider the distance between two adjacent nodes as 1. Furthermore, the estimated distance between two nodes,  $n_a$  and  $n_b$ , is the Manhattan distance and is defined by  $|x_a - x_b| + |y_a - y_b|$ .

The outline of a possible implementation is as follows. Define a class that models nodes and their containing information: e.g., the parent and distances information of a node. In addition, define an array of node object points that is of the same dimensions as the char array (our environment representation). This array effectively models our graph. This array of node points is initially empty—that is, it contains only NULL pointers—except for a start node. Next, your program starts adding new nodes to the graph, according to the rules of the A\* algorithm, until either the goal is reached or the entire environment is explored. One additional note: it is not necessary to maintain sorting of open nodes based on their  $f$ -values; it is fine to check all open nodes for each A\* loop.

### The Output

The third and last step is quite similar to initial import step. If A\* has found a path that connects ‘S’ with ‘G’, we write the original environment plus the path information to a new text file. The path can be added to the text file by replace the ‘ ’ characters contained in the path by ‘.’ characters, as illustrated in figure 2. If no path exists that connects ‘S’ with ‘G’, then report this in the output file instead of the map plus path.



```

*****
*          ****. .... ** . . . . ****
* . . . . . ****. . . . . ****. *
* . ****. . . . . ****. * G. *
* . * . . . . . ****. . . . .
* . * . . . . . * * * * *
* . ****. . . . . ****. * * * * *
* . . . * . . . . . ****. . . . .
* . . . * . . . . . ** * * * * *
****. * . . . . . * * * * *
* S . . . . . **** * * * * *
*****

```

Figure 2: Left: Allowed movement over grid. Right: Example of how the output file should look.

**Submitting your Work**

The submission deadline is Monday (22th of Feb.) at midnight. Email your zipped *solution* to [basten@cs.uu.nl](mailto:basten@cs.uu.nl). Please do not forget to add your names and student numbers.

The assignment will be graded as ‘satisfactory’ when the following requirements are fulfilled:

- It is not allowed to use STL vectors and strings. The goal of this assignment is to get used to heaps and arrays. (You will need to declare the array on the heap, for we do not know the size of the floorplan in advance)
- Make sure all the variables on the heap are properly destroyed at the end of the program. Double check that the destructors are invoked when necessary.
- The floorplan should be read from a file.
- The result should be written to a file (including reporting when no path exist).
- The A\* algorithm needs to be correctly implemented.
- Design your code in an object-oriented fashion. So, use classes and inheritance where appropriate.
- Please include sufficient *comments* to the code so that at least the role of each function and class in your program is clear.
- I need to simply open the solution, compile and run the contained project, and check the output. If it does not compile and run in both debug and release mode, there will be no grading.
- In addition, please make sure to ‘Clean’ your solution, and delete the ‘.ncb’-file and ‘.ilk’ files. A Visual Studio solution consist of the solution file (.sln), the project file (.vcproj) and the source files (.cpp and .h).