# Homework 6

*Due: 19 Oct 2007*

## Problem 6.1 (×4)

In this project you're going to write a route finder. The applet you write should be able to read in map files, and using a variety of different heuristics to shape the search, display the shortest path between the start and finish squares, if any. (If there are multiple equally-short paths, any one of them can be displayed.)

The input format and GUI are as described in Lab 6.

The algorithm is the A algorithm as discussed in lecture. It should be efficient. For your first pass, you can assume a default heuristic of $h(X) = 0$, but see below for other heuristics you'll need to be able to run with. Remember that your implementation will just do one step of the algorithm at a time, rather than implement as a `while` loop. (The code provided for Lab 6 already meets these requirements; just make sure not to break them as you edit!)

When the algorithm finishes, the interface should clearly indicate whether there is, or is not, a path, and if there is, you should reconstruct it and draw it, also as described in Lab 6.

NB: as you implement the following subproblems, you don't have to retain the various interfaces used in the existing code base. If you need to add new methods or change headers, feel free, as long as the `.html` file works.

### Heuristics

Confirm that everything works with a simple queue-based agenda before trying to implement the heuristics; once you're ready, though, you should implement agendas whose policies are organised according to each of the following four heuristics:

- $h(X) = 0$
- $h(X) = E(X) =$ Euclidean distance between $X$ and the finish, truncated to next lower integer

- $h(X) = M(X) =$ Manhattan distance between $X$ and the finish
- $h(X) = P(X) =$ "Proximity sensor metric" to finish:
  - $P(X) = M(X)$ if$M(X) < 8$
  - $P(X) = 8$ if$M(X) \geq 8$

The GUI should let a user start the search with any of the four metrics, either using four separate Start buttons or by adding a pull-down list.

### Heuristic tiebreakers

Once you've implemented all the heuristics and have tried them on a variety of test cases, you may have noticed that even with an informed heuristic, the algorithm still is exploring a lot of space it "obviously" shouldn't have to. Resolving this flaw is not trivial, but neither is it especially difficult: it involves retooling your agenda policy so that if the main utility function $f(X) = g(X) + h(X)$ computes equal utility for two different locations, you use a modified utility function to break the tie. Implement this; add a checkbox to the GUI, and if it is checked when the solver starts, make your agenda policy break ties when the utility function turns up equal values.

### Teleportation

Add another square type to the input, represented by the character `'@'`. If present in the input at all, there will be exactly two of them; they represent teleporters. A teleporter behaves exactly like an open square, with one exception: from one teleporter, the other teleporter is exactly one step away (just as if they were adjacent). A run given the input

```
13 3
......#......
.@.o..#..*.@.
......#......
```

should successfully find a path from start to finish via the teleporters. Furthermore, since each teleporter is just one step away from the other, the input

```
13 3
.............
.@.o.....*.@.
.............
```

should still find a path via the teleporters—that route has four steps exclusive of the endpoints, while the non-teleporter shortest route has five.

Note that teleportation is not obligatory; a run on the input

```
13 3
.............
..o.@.*...@..
.............
```

should find a shortest path that cuts right across the first teleporter without involving the second one at all.

## Problem 6.2 (×2)

The previous problem was all about the implementation. This one is more analytical; you could theoretically get full credit for this problem without writing any part of the implementation. (This is not recommended.)

In your electronic handin, include an appropriate number of test maze files. They should both test the algorithm for correctness and highlight the strengths and weaknesses of the different heuristics. Include comments in the files as to which of these goals are being accomplished.

The squares that are shaded grey—i.e. those that were discovered in the course of the search—collectively show the "shape" of a search, and this shape is affected by which heuristic was used. As a separate paper handin or in a plain text file in your electronic handin, select a test maze where all four heuristics yield different search shapes. Explain why the shapes look the way they do.

On paper or in a plain text file, discuss how the existence of teleporters affect admissibility of the various heuristics. For any that remain admissible, sketch a proof as to why. For those that do not, give an example input that "breaks" it.

On paper or in a plain text file, discuss how the heuristic tiebreaker affects admissibility of the various heuristics.