AI Programming (IT-3105) Project Module #1: Implementing and Testing the A* Algorithm on a Navigation Task

Purpose: Gain hands-on experience with best-first search.

1 Introduction

In this assignment, you must first become familiar with the A* algorithm by programming it yourself, **from scratch**, in the language of your choice. It is recommended that you consult the accompanying document entitled "Important Details of the A* Algorithm", (IDAA) as it contains several important implementation details that you may not find in a standard textbook or online description of A*.

The A* algorithm was originally designed for finding the shortest path from start to goal for the mobile robot named *Shakey*, built at the Stanford Research Institute in the late 1960's. The basic concepts were then easily extended to many search problems: those in which a) solutions are gradually pieced together by the problem solver, and b) the distance between a partial solution and a final solution can be estimated (by a *heuristic* function).

Navigation problems from point A to B are very graphic illustrations of an A*-friendly situation; and finding a proper heuristic is generally not too difficult. For these problems, each state of the search space is a path from A to some other point on the map, and expanding a state involves adding one more step to its path. A goal state is one whose path begins with A and ends with B.

2 General Requirements

For this assignment, you will use simple 2-dimensional grids on which to navigate. A problem specification will include:

1. the dimensions of the grid, e.g. 4-by-5
2. the location of the start point, e.g. cell (0,0)
3. the location of the goal point, e.g. cell (3,4)
4. the locations of each rectangular obstacle, along with its width and height

Figure 1 shows one such grid along with its specification.

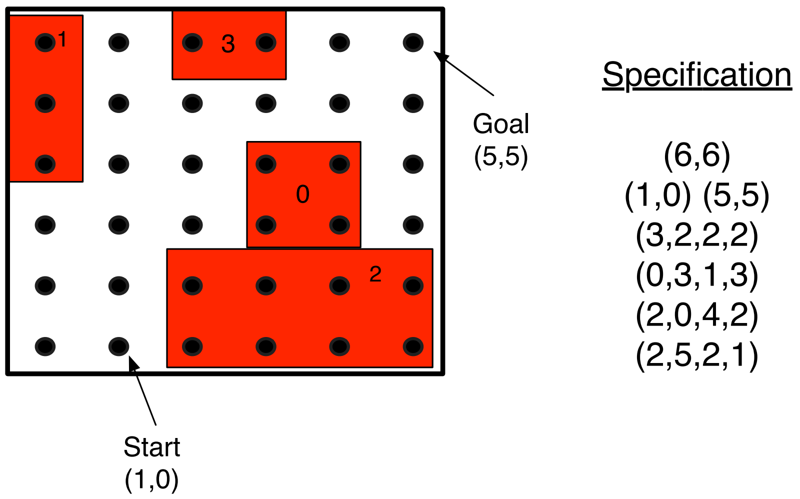

Figure 1: A 2-dimensional grid for a navigation task, along with its specification, which begins with the grid's dimensions followed by the locations of the start and goal cells. The remaining quads specify the locations and sizes of each obstacle, where the first two values specify the x and y coordinates of the rectangle's lower left corner, while entries 3 and 4 specify the width and height, respectively. Note the use of zero-based indexing of grid cells and obstacles. All dots in the figure represent the center of a grid cell. The origin of the grid, cell (0,0), is at the bottom left.

Your code will convert this specification into an actual grid, with obstacles, and then A* will find the shortest path from start to goal.

A legal move is one cell in the horizontal or vertical direction from the current cell. You can add a diagonal move to one of the four diagonal neighbors if you'd like, but this must be clearly documented in your report. The walls of the grid are real barriers: there is no wrap-around. In other words, a path cannot move left from (0,0) and end up in cell (9,0) (for a grid with 10 columns).

Your code will also need to include a GUI that displays the grid and a series of paths corresponding to the search nodes popped from your **agenda** data structure – as discussed in IDAA and as demonstrated by the instructor in class. In other words, your GUI will display the path corresponding to each search node as it is popped from the agenda.

For each specification, a **navigation test** will consist of the following subtasks:

1. A run, with GUI visualization, of A* in best-first mode.
2. A run, with GUI visualization, of A* in depth-first mode.
3. A run, with GUI visualization, of A* in breadth-first mode.
4. A comparison of the total number of nodes generated for these 3 runs.
5. A comparison of the lengths of the solution paths generated for these 3 runs.

See IDAA for the simple modifications needed to get depth-first and breadth-first behavior from the A* algorithm.

Your program must accept scenario specifications either from the command line or from a file. It should NOT be necessary to type these specifications into your source code, re-compile your code, etc., just to test a different scenario. Failure to satisfy this simple requirement can cost several points during the demo session. Good code should never need to be recompiled just to handle a new input.

It is ok to set some large upper limit (depending upon your computing power) on the total number of generated nodes in order to avoid VERY long runs. Thus, your version of A* would just stop after reaching that limit. Some scenarios give much worse behavior for one or more of the 3 search alternatives: depth-, breadth- and best-first, and this can be verified by simply showing that some alternatives reach the upper node limit, while others do not. However, such a limit should not be necessary for the scenarios of Table 1, if you implement A* correctly.

Table 1: Several scenarios for testing your A* algorithm

Navigation Scenarios				
Index	Dimensions	Start	Goal	Barriers
0	(10, 10)	(0, 0)	(9, 9)	(2, 3, 5, 5) (8, 8, 2, 1)
1	(20, 20)	(19, 3)	(2, 18)	(5, 5, 10, 10) (1, 2, 4, 1)
2	(20, 20)	(0, 0)	(19, 19)	(17, 10, 2, 1) (14, 4, 5, 2) (3, 16, 10, 2) (13, 7, 5, 3) (15, 15, 3, 3)
3	(10, 10)	(0, 0)	(9, 5)	(3, 0, 2, 7) (6, 0, 4, 4) (6, 6, 2, 4)
4	(10, 10)	(0, 0)	(9, 9)	(3, 0, 2, 7) (6, 0, 4, 4) (6, 6, 2, 4)
5	(20, 20)	(0, 0)	(19, 13)	(4, 0, 4, 16) (12, 4, 2, 16) (16, 8, 4, 4)

3 Deliverables

1. A 3-page report describing the central aspects of your A* program, such as the agenda and how it is managed. This must clearly and concisely document the generality of your program: it should illustrate how your program can easily be reused on other search tasks, with only a little subclassing needed for each new task. So the core code should NOT have details specific to the navigation task. The report must also describe your heuristic function and your method for generating successor states. The report must NOT exceed 3 pages, including all diagrams, code segments, etc. (**5 points**)
2. A demonstration of your program performing *navigation tests* for any of the scenarios of Table 1 along with a few other scenarios (of the same format) that will be given to you during the demo session. For this demonstration, your GUI must clearly show paths, and each run must end by printing the total number of search nodes created during the run, along with the length of the solution path. (**10 points**).
3. Well-commented source code for the general A* algorithm. (**0 points**)
4. Well-commented source code for the specializations of A* needed to handle the navigation task. (**0 points**).

A zip file containing your report along with the commented code must be uploaded to It's Learning prior to the demo session in which this project module is evaluated. You will not get explicit credit for the code, but it is crucial that we have the code online in the event that you decide to register a formal complaint about your grade (for the entire course).

The 15 total points for this module are 15 of the 100 points that are available for the entire semester.

The due date for this module and a second module is the first demo day.