|  |  |  |
| --- | --- | --- |
|  |  | **CS560 Fall 2015**  **Assignment 1: Rat in a Maze**  **Due date: Thurs, September 10, 11:59:59PM** |

In this assignment, you will implement general search algorithms and apply them to solving mazes.

You are free to use any high-level programming language you are comfortable with and to work in groups of up to 3 people.

### 1. Solving Mazes with Search

Consider the problem of finding a path through a maze for your rat from a given start state to a given goal state using search. The maze layout will be provided in a simple text format, where '%' stands for walls, 'S' for the starting position, and 'G' for the goal. In this problem the rat is allowed to move only horizontally or vertically (not diagonally) and all step costs are equal to one.

Implement the following search algorithms for solving mazes:

* Depth-first search
* Breadth-first search
* Greedy best-first search
* A\* search

As mentioned in class, these algorithms only differ in how they manage the frontier. In your implementation, make sure you get all the bookkeeping right. This includes handling of repeated states (in particular, what happens when you find a better path to a state already on the frontier) and saving the optimal solution path.

For greedy and A\* search, use the Manhattan distance from the current position to the goal as the heuristic function.

Run each of the above algorithms on the following inputs:

* [Small Maze](http://www.tamaraberg.com/teaching/Fall_15/HW/HW1/smallMaze.txt)
* [Medium maze](http://www.tamaraberg.com/teaching/Fall_15/HW/HW1/mediumMaze.txt)
* [Big Maze](http://www.tamaraberg.com/teaching/Fall_15/HW/HW1/bigMaze.txt)

For each maze and each search algorithm, include the following in your report:

* The solution, displayed by putting a '.' in every maze square visited on the path ([example solution](http://www.tamaraberg.com/teaching/Fall_15/HW/HW1/smallMazeSolution.txt) to the small maze).
* The path cost of the solution (number of steps taken to get from the start state to the goal state).
* Number of nodes expanded by the search algorithm.

### 2. Designing “difficult” mazes

Design two different mazes: one that is much more "difficult" for A\* search than for greedy best-first search with Manhattan distance heuristic, and one that is the opposite. Here, we measure difficulty by the number of nodes expanded, not by the cost of the path found: that is, an input is "easier" for greedy search if it finds a solution by expanding fewer nodes than A\*, even if the solution is suboptimal.

For each of your two mazes and search algorithms, display the computed solution paths, and report the solution costs and numbers of nodes expanded for each algorithm. The greater the disparity in the number of nodes expanded on each input by the two algorithms, the better. Discuss what makes each input "hard" or "easy" for the respective algorithms.

### 3. Eating all the cheese

Now your goal is to find the shortest path through a maze to eat all the cheese! Once again, the rat starts at position ‘S’, but now there is no end goal position. Instead, the goal is achieved whenever the rat manages to eat all the cheese (denoted as dots in the maze). Whenever you move into a location with a piece of cheese, it becomes eaten. Assume all step costs are equal to one.

Revise your A\* search code from Part 1 to deal with this scenario. This will require changing the goal test (have you eaten all the cheese?) and the state representation (besides your current position in the maze, is there anything else you need to know?). Run your A\* search on the following inputs:

* [Small cheese](http://www.tamaraberg.com/teaching/Fall_15/HW/HW1/smallcheese.txt)
* [Tricky cheese](http://www.tamaraberg.com/teaching/Fall_15/HW/HW1/trickycheese.txt)
* [Medium cheese](http://www.tamaraberg.com/teaching/Fall_15/HW/HW1/mediumcheese.txt)
* [Big cheese](http://www.tamaraberg.com/teaching/Fall_15/HW/HW1/bigcheese.txt)

For this part of the assignment, it is crucial to design a good heuristic. For full credit, your heuristic should be admissible and should permit you to find the solution for the medium and big searches in a reasonable amount of time. If you cannot find the optimal solutions for these two inputs, you can still get most of the credit by using a non-admissible heuristic or a sub-optimal search algorithm. In your report, explain the heuristic and the algorithm you chose, and discuss whether it leads to an optimal solution.

In your report, for each maze, give the solution cost and the number of nodes expanded. For the first two mazes, show your solution by numbering the goals in the order in which you reach them (once you run out of numbers, use lowercase letters). For the last two mazes, there is no need to display your solution, unless you want to come up with a nice animation for extra credit.

**Submission Instructions**

HW should be uploaded to the classroom.cs.unc.edu server (one submission per group uploaded into the directory of one of the group members). Email Ric ([poirson@cs.unc.edu](mailto:poirson@cs.unc.edu)) with the location of your HW submission and your group member names. All submitted code must run on the classroom server for full credit. **Note**, if you edit any portion of your HW after the submission date that time-stamp will become your submission time and count into your late days.

Submissions should include:

1. A report in PDF format called HW1.pdf with the names of all group members indicated on top. The report should describe your implemented solution and fully answer the questions for every part of the assignment. Your description should highlight the most "interesting" aspects of your solution, i.e., any non-obvious implementation choices and parameter settings, and what you have found to be especially important for getting good performance. Feel free to include pseudo-code or figures if they are needed to clarify your approach. Your report should be self-contained and it should (ideally) make it possible for us to understand your solution without having to run your source code.

All group reports need to include a brief statement of individual contribution, i.e., which group member(s) was/were responsible for which parts of the solution and submitted material.

2. Your source code and a README file indicating how to run your code. Code should be well commented and it should be easy to see the correspondence between what’s in the code and what’s in the report.

**Late policy**: Assignments are due at 11:59pm on the assigned date. Students have 5 free late days to use over the course of the semester (e.g. an assignment submitted at 2am after the deadline will count as 1 day late). After free late days are used, assignments will be accepted up to 1 week late at a penalty of 10% per day.

**Academic integrity:** All code and written responses for assignments should be original within your group. To protect the integrity of the course, we will be actively checking for code plagiarism (both from current classmates and the internet).