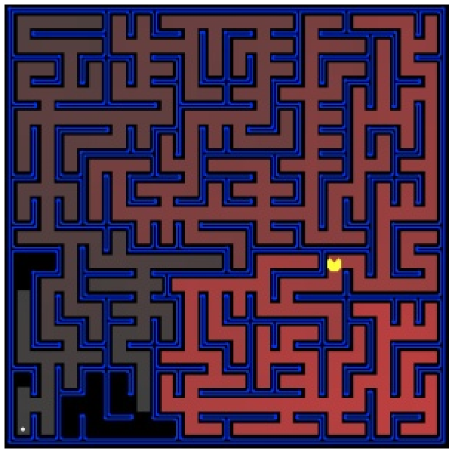

# CSCI 446 – Artificial Intelligence/Fall 2018

# Assignment 1: Search

**Due date: Friday, Oct 8, Midnight (11:59pm)**

Credits: Berkeley CS188 Pacman projects, Kyo Hyun Kim and Krishna Kothapalli

In this assignment, you will build general-purpose search algorithms and apply them to solving puzzles. You will be in charge of a "Pacman" agent that needs to find paths through mazes while eating one or more dots or "food pellets."

As stated in the beginning of the course, **you are free to use any high-level programming language you are comfortable with**. This includes (but is not limited to) Java, C++, and Python. The focus of this course is on problem solving, not programming, and the grading will primarily be based on the quality of your solutions and your analysis, as evidenced by your **written report**.

**You have the option of working in groups of up to three people.** Needless to say, working in a group will not necessarily make your life easier, as the overhead of group coordination can easily outweigh the benefits.

## Description

### 1.1 Basic pathfinding

Consider the problem of finding the shortest path from a given start state while eating one star or "food pellet" The image at the top of this page illustrates the simple scenario of a single food pellet (dot), which in this case can be viewed as the unique goal state. The maze layout will be given to you in a simple text format, where '%' stands for walls, 'P' for the starting position, and '*' for the food pellet/star. All step costs are equal to one.

Implement the state representation, transition model, and goal test needed for solving the problem. For the state representation, besides your current position in the maze, is there anything else you need to keep track of? Next, implement a unified top-level search routine that can work with all of the following search strategies, as covered in class:

- Depth-first search;
- Breadth-first search;
- Greedy best-first search;
- A* search.

For this part of the assignment, use the Manhattan distance from the current position to the goal as the heuristic function for greedy and A* search.

Run each of the four search strategies on the following inputs:

- Medium maze;
- Big maze;
- Open maze.

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

- The solution, displayed by putting a '.' in every maze square visited on the path.
- The path cost of the solution, defined as the number of steps taken to get from the initial state to the goal state.
- Number of nodes expanded by the search algorithm.

**Tips**

- 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. These topics have been extensively covered during the lectures.

- Pay attention to tiebreaking. If you have multiple nodes on the frontier with the same minimum value of the evaluation function, the speed of your search and the quality of the solution may depend on which one you select for expansion.

- Make sure you implement a unified top-level search algorithm that can take each of the four strategies as special cases. In particular, while DFS can be implemented very compactly using recursion, we want you to avoid this approach for the sake of the assignment (among other things, you can much more easily exceed the maximum depth of the recursion stack than if you explicitly represent the frontier as a stack).

- You will be graded primarily on the correctness of your solution, not on the efficiency and elegance of your data structures. For example, we don't care whether your priority queue or repeated state detection uses brute-force search, as long as you end up expanding (roughly) the correct number of nodes and find the optimal solution. So, feel free to use "dumb" data structures as long as it makes your life easier and still enables you to find the solutions to all the inputs in a reasonable amount of time.

## Report Checklist

Your report should briefly describe your implemented solution. Your description should focus on the most "interesting" aspects of your solution, i.e., any non-obvious implementation choices (including programming language chosen) and parameter settings, and what you have found to be especially important for getting good performance. Feel free to include pseudocode (DO NOT PUT CODE IN YOUR REPORT) 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. For full credit, in addition to the algorithm descriptions, your report should include the following

For every algorithm (DFS, BFS, Greedy, A*) and every one of the three mazes (medium, big, open): give the maze with the computed path, the solution cost, and the number of expanded nodes (12 cases total).

**Statement of individual contribution:**

- All group reports need to include a write up from each group member on what you were responsible for and what you found hard, interesting, unexpected, etc. We reserve the right to contact group members individually to verify this information.

***WARNING: You will not get credit for any solutions that you have obtained, but not included in your report!*** For example, if your code prints out path cost and number of nodes expanded on each input, but you do not put down the actual numbers in your report, or if you include pictures/files of your output solutions in the zip file but not in your PDF. The only exception is animated paths (videos or animated gifs).

## Submission Instructions

By the submission deadline, **one designated person from the group** will need to upload the following to D2L/Brightspace:

1. A **report** in **PDF format**. Be sure to put the **names** of all the group members at the top of the report. The name of the report file should be **lastname_firstname_a1.pdf** (based on the name of the designated person).

2. Your **source code** compressed to a **single ZIP file**. The 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. You don't need to include executables or various supporting files (e.g., utility libraries) whose content is irrelevant to the assignment. If we find it necessary to run your code in order to evaluate your solution, we will get in touch with you.

   The name of the code archive should be **lastname_firstname_a1.zip**.

Multiple attempts will be allowed but only your last submission before the deadline will be graded. **We reserve the right to take off points for not following directions.**

**Late policy:** You must submit by the full package (report and source code). No exceptions.