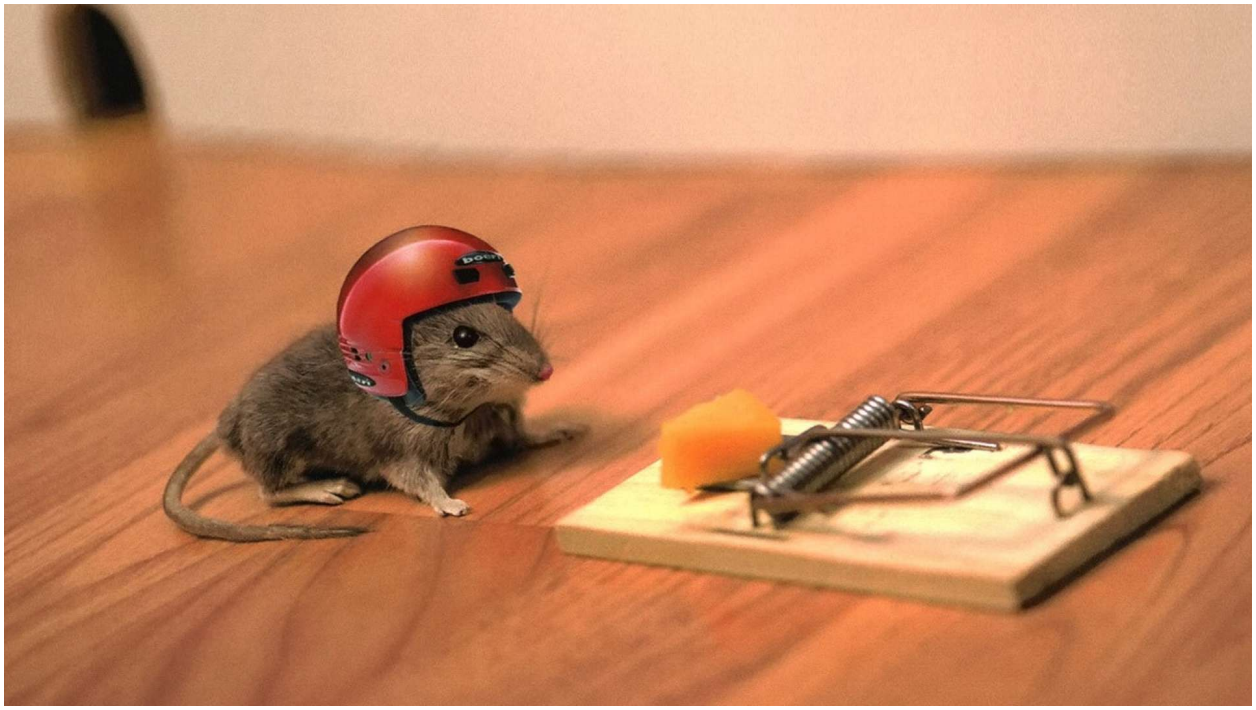
CS365 Lab A - Search and Pathfinding

This lab has been adapted from Svetlana Lazenbik's work.



This Lab is worth 60 points. Please follow the Notes for All Labs carefully. The lab should be done in **Python 3**.
This is a pair assignment. You and your partner should work together for all portions of the assignment, unless otherwise indicated. You should not split it up and each do half. This lab is a significant amount of work, and will overlap with reading assignments and short discussion assignments. If this is your first upper-level CS class, this assignment is likely to be an adjustment in terms of expectations. Plan accordingly.

If you have any questions about the lab, please ask over email, in class, or on Piazza.

Your **deliverables** are:

- Due January 30th:
    - Design file, as a .pdf.
    - The design file has its own submission area on Moodle.
- Due February 7th:
    - One or more code files
    - A README
    - A .pdf report file

    - If you wish to write a script that tests your code, you can also submit that. Indicate how to use it in the README.

We will spend time in class on **January 28th** discussing Part 1. You should begin work on Part 1 before this, so you will be able to participate in the discussions.

- Complete Part 1 and ideally Part 2 by **January 30th.** (You should at least make solid progress on Part 2 by this date.) We will spend class on **January 31st** discussing your choices for these parts. The only thing due for submission on January 30th is the **Design File**. The design file has its own submission area on Moodle.
- We will spend part of **February 4th**'s class discussing Parts 2, 3 and 4. You should make as much progress as possible on these parts before February 4th, so you will be able to ask questions about things you are stuck on.
- The full lab is due **February 7th.**

The files for this lab are located in ~barbeda/cs365-share/lab-a

The problems in this lab all involve an agent traversing a maze. The agent's goal is to collect all of the prizes. (There is no exit; it is complete after all of the prizes are collected.) The agent can move one square at a time in any of the cardinal directions. It cannot move diagonally. We can assume that the agent automatically collects a reward when it enters the tile that contains the reward. There is not a separate action for picking the reward up.

The maze files are simple text files. The symbols in the maze files mean the following:
P is the starting position for the agent;
% is a wall; the agent cannot enter this tile
. is a reward
Space is an open tile

You may wish to view the maze files in the /lab-a directory before you begin thinking about the lab.

As part of this lab, you will need to write code that supports working with your environments. At a minimum, you will likely need some way to read in and parse the maze files.

You should read Chapter 3 before beginning the lab. Several portions of the lab rely directly on it.

Part 1)

Part 1 should be completed by **January 30th**. The only part that needs to be submitted on the 30th is part D, the design file.

Read the entire lab before beginning. Your implementation choices depend on each other, and the choices made in this part might make later parts harder or necessitate revisiting your implementation. Several of these terms are defined in section 3.1.1.

A) Implement a *state representation scheme*. This is a representation of the current state of the world. Think about what we might need to keep track of to understand what the world is like. Remember that there might be multiple prizes in the maze.
Hint: Start by making a list of all of the things we need to keep track of.
Hint: Simpler is often better. Don't choose a complex state representation scheme unless there are strong reasons for doing so.

B) Implement a *transition model*. The four actions the agent can take are North, South, East, and West. The step cost of all of these actions is 1.
Hint: You may want to test out your transition model using a simple maze.

C) Implement a *goal test*. This should be a function that takes a state representation as an argument and returns **True** if the goal is met and **False** otherwise. You should have a function that I can call that does this, and an example of how to call it in your **README** file. Think carefully about this, as your definition needs to account for the fact that if there are multiple prizes then the agent needs to collect all of them.

D) Write a short **design file** (~1 page) describing and defending your implementation choices for Part 1 of the lab and your implementation plans for Parts 2, 3, and 4 of this lab. For example, you might discuss why you chose to represent environments as two-dimensional arrays, or why you chose to chose to represent the actions types as strings.
This should be submitted as a .pdf on **January 30th**. Your design file should make it clear that you have a solid understanding of the algorithms being used. (You are allowed to change your designs later based on class discussion; you do not have to stick with the plans you submit.)

Part 2)
Attempt to complete Part 2 by **January 30th**. Even if you aren't totally satisfied with your solution or you get stuck, making as much progress as possible on this will set you up for class on the 31st.

For this part of the lab and Part 3, you will be using the 1prize mazes. Each of these has exactly one prize in them.

Implement **Depth-first Search** for the problem space where there is only one prize. If you do not feel comfortable with this algorithm and how it works, be sure to seek clarity early.

Your API should have a function named:

single_dfs

This function should take one argument: a file location containing the maze. (You may want one or more helper functions in addition to this.)

Your **README** should include examples of how to run each of these functions.

When run, your function should display each of the following in a clean and readable way:

- 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.
- The number of nodes expanded by the search algorithm.

In your **report,** include the results of running the function on each of the three 1prize maze files. (This is three results in total.) Make sure to use a fixed-width font (such as Courier or Source Code New) when displaying the solutions, and ensure that they are still readable after saving to .pdf.

Part 3)

A) Implement each of the following algorithms for the problem space where there is only one prize. If you do not feel comfortable with what these algorithms are or how they work, be sure to seek clarity early.

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 best-first and A* search. (You will design your own heuristic for A* during part 3.)

Your API should have three additional functions named:

single_bfs
single_gbfs
single_astar

Each of these three functions should take one argument: a file location containing the maze, just as single_dfs does. (You will likely want several helper functions in addition to these.)

Your **README** should include examples of how to run each of these functions.

When run, your functions should display each of the following in a clean and readable way:

- 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.
- The number of nodes expanded by the search algorithm.

In your **report,** include the results of running each of the functions on each of the three 1prize maze files. (This is nine additional results, in addition to the three generated during Part 2.) Make sure to use a fixed-width font (such as Courier or Source Code New) when displaying the solutions, and ensure that they are still readable after saving to .pdf.

The optimal path costs for each maze are as follows. Note that an algorithm that is not guaranteed to produce an optimal path may find a path longer than these:

45 Open
94 Medium
148 Large

Part 4)

For this part of the lab, you will be using the multiprize mazes. Each of these has a larger number of prizes in them. The agent must pass over ALL of the prizes; after passing over the last prize, the goal state is achieved. The prizes can be collected in any order. Make sure that you understand how A* works; a system that selects the nearest prize, does A* to path to it, then selects the next-nearest prize is *not* doing A* search.

Your state representation, goal test, and transition model should already work with this setup, but if they do not, update them now. The next challenge is to solve the multiprize mazes using A* search. You will construct an admissible heuristic of your own design.

You should be able to handle the multiprize-tiny.txt search using uninformed BFS - and in fact, it is a good idea to try that first for debugging purposes, to make sure your representation works with multiple dots. (You don't need to submit anything related to using BFS on the multiprize ones, however.) However, to successfully handle all the inputs, it is crucial to come up with a good heuristic. For full credit, your heuristic should be admissible. *Ideally, your search should permit you to find the solution for the medium search in a reasonable amount of time. (If your search takes more than ten minutes or so, note this in your report.)*

Update 02/08/20: Your search needs only to run on multiprize-micro.txt in a reasonable amount of time. You may want to test your search on the other multiprize mazes to see how it handles them, but only timely success on multiprize-micro is required.

In your **report**, explain the heuristic you chose, and discuss why it is admissible and whether it leads to an optimal solution. If you believe the Manhattan Distance function you implemented in Part 3 is appropriate, you can continue to use that.

Your API should have a function called

multi_astar

That takes one argument, as above. It should display:

- The solution, by showing the maze, but with all of the rewards changed into numbers, starting with 0, in the order that they are visited. If there are more than 10 rewards, start using lowercase letters, then uppercase letters. The path that is followed will not be reflected in your solution, just the order that the rewards are collected.
- The path cost of the solution, defined as the number of steps taken to get from the initial state to the goal state.
- The number of nodes expanded by the search algorithm.
- To help you check your work, the minimum path costs for each of the provided mazes are:

Micro: 21 Tiny: 36
Small: 143
Medium: 207

Include the results of running your function on each of the three multiprize files in your **report**. If it takes too long to run on the larger mazes, note that in your report. (If it takes too long to run even on the smallest one, you may need to optimize things differently.)

Shallow and Deep Copying

Depending on your representation choices, it is easy to run into trouble on this lab by creating many references to the same object. In Python, if there are multiple references to an object, and that object's properties change, they will change for all references to the object. An easy way to see this is by running this code:

```
a = [1,2]
b = a
b[0] = 3
print(a)
```

You will need to be careful that not all references to your representations are to the same object. Read here: copy — Shallow and deep copy operations — Python 3.8.1 documentation for more information.