Part 1: Navigation

As those of you in the online section learned in Module 1, a certain autonomous agent likes to fly around the house and interrupt video recordings at the most inopportune moments (Figure 1). Suppose that a house consists of a grid of $N \times M$ cells, represented like this:

```
....XXX
.XXX...
....X..
.X.X...
.X.X.X.
pX...X@
```

As you can see, the map consists of N lines (in this case, 6) and M columns (in this case, 7). Each cell of the house is marked with one of four symbols: `p` represents the agent's current location, `X` represents a wall through which the agent cannot pass, `.` represents open space over which the agent can fly, and `@` represents your location (presumably with video recording in progress).

Figure 1: The autonomous agent, after a bath.

Your goal is to write a program that finds the shortest path between the agent and you. The agent can move one square at a time in any of the four principal compass directions, and the program should find the shortest distance between the two points and then output a string of letters (L, R, D, and U for left, right, down, and up) indicating that solution. Your program should take a single command line argument, which is the name of the file containing the map file. For example:

```
[<>djcran@silo ~] python3 route_pichu.sh map1.txt
Shhhh... quiet while I navigate!
Here's the solution I found:
16 UUURRDDDRRUURRDD
```

You can assume that there is always exactly one `p` and one `@` in the map file. If there is no solution, your program should display path length `-1` and not display a path.

To help get you started, we have provided some initial code that is already available in your GitHub repo. Here's what to do to complete the program.

1. We've already created a GitHub repository for you for this assignment. You can see the name of your repository by logging into IU Github, at `http://github.iu.edu/`. In the upper left hand corner of the screen, you should see a pull-down menu. Select `cs-b551-fa2021`. Then in the box below, you should see a repository called *youruserid*-a0, (If you do not see `cs-b551-fa2021` or a repository with your userid, it probably means that did not log into GitHub to create your account during the A Few Action Items activity during week 1 of class, so we were not able to add you to a team. Post a private message on Q&A Community so that we can create your repo manually.)

   To get started, from the SICE Linux machines, clone the github repository:

   `git clone git@github.iu.edu:cs-b551-fa2021/`*`your-repo-name`*

   If that doesn't work, instead try:

   `git clone https://github.iu.edu/cs-b551-fa2021/`*`your-repo-name`*

   where *your-repo-name* is the one you found on the GitHub website above. (If neither command works, you probably need to set up IU GitHub ssh keys. See Canvas for help.)

2. Now you should see a program called `route_pichu.py`. We have also provided two sample map files, `map1.txt` and `map2.txt`. You can run our program like this:

   ```
   python3 route_pichu.py map1.txt
   ```

   Unfortunately, the program does not work very well; it will probably enter an infinite loop and you'll have to press CONTROL-C to kill it. Nevertheless, the code is a good starting point, so familiarize yourself with it. Figure out the precise search abstraction that the program is using and include it in your report. In other words, what is the set of valid states, the successor function, the cost function, the goal state definition, and the initial state?

3. Why does the program often fail to find a solution? Implement a fix to make the code work better, and explain what you did in the report.

4. Complete the program so that it finds and displays the correct solution. Check the starter code comments for specifications on the search() function, because our autograder (and the pytest command above in ***Coding Requirements***) calls it directly.

Part 2: Hide-and-seek

Suppose that instead of a single agent as in Part 1, you have adopted k agents. The problem is that these agents do not like one another, which means that they have to be positioned such that no two agents can see one another. Write a program called `arrange_pichus.py` that takes the filename of a map in the same format as Part 1 as well as a single parameter specifying the number k of agents that you have. You can assume $k \geq 1$. Assume two agents can see each other if they are on either the same row, column, or diagonal of the map, and there are no walls between them. An agent can only be positioned on empty squares (marked with .). It's okay if agents see you, and you obscure the view between agents, as if you were a wall. Your program should output a new version of the map, but with the agents' locations marked with `p`. Note that exactly one `p` will already be fixed in the input map file. If there is no solution, your program should just display `False`. Here's an example on the same sample output on the same map as in Part 1:

```
[<>djcran@silo ~] python3 arrange_pichus.py map1.txt 5
....XXX
.XXXp..
.p..X..
.X.X...
.X.X.Xp
pX.p.X@
```

We've again given you some code to get started with, but it's not fully working; the configurations it finds often allow agents to see one another, and it can be quite slow. Fix the code so that it works, and then try to make it run as quickly as possible. In your report, explain the search abstraction you've used – what is the state space, initial state, goal state, successor function, and cost function?

Make sure to test your program on maps other than the one we've given, including ones of different sizes and shapes. Check the starter code comments for specifications on the solve() function, because our autograder (and the pytest command above in ***Coding Requirements***) calls it directly.