Project 2

Due: Tuesday, December 9th 11:59pm

Description

In this project you will write a prolog program that can solve a maze, creating a predicate `find_exit/2` for the user to query. As a reminder the format `f/N` represents a predicate with functor `f` and arity `N`. **Do not forget to follow the general instructions – maintaining a git repository and devlog.**

Details

You will write a predicate `find_exit/2`. The first parameter is the maze as described below, and the second should be a list of actions that will solve the maze. The predicate succeeds if following the list of actions will lead from the start space to any exit space. The predicate must work even when called with an unbound variable for its second parameter. The predicate should fail if the maze is invalid, if following the actions does not end in an exit space, or if an action would result in moving off the maze or onto a wall.

The Maze

A maze is a list of rows with each row being a list of cells. Each cell can be a `f`, `w`, `s`, or `e`.
- An `f` represents a floor space (empty).
- A `w` represents a wall.
- An `s` represents the start space. There should be exactly one start space.
- An `e` represents an exit. There can be more than one exit.

The Actions

An action can be `left`, `right`, `up`, and `down`. When solving the maze, the coordinates start on the start space.
- A `left` action moves the coordinates to the previous column.
- A `right` action moves the coordinates to the next column.
- A `up` action moves the coordinates to the previous row.
- A `down` action moves the coordinates to the next row.

When changing rows or columns, the other dimension does not change.

Some Tips

- Don't forget that a predicate can have more than one rule.
- Take advantage of unification, and try not to overcomplicate your solutions.
- You can use negation-as-failure and the cut to simplify some predicates

- When using the command-line, you can load multiple files. Use the included files to help with testing.
  - The included `example.pl` file contains some simple example mazes and a predicate to pretty print a maze.
  - The included `test.pl` provides `gen_map/4`. The first parameter should be an integer greater than zero. Four is a good value. It controls the complexity of the maze. The second and third parameters are the number of rows and columns (resp.) for the maze you want to generate. The fourth parameter will unify with the generated maze.