Search Mini-Assignment

In this assignment, you will program some search algorithms and apply them to solve a path-finding problem. 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 dot(s) (see sample maze file). All step costs are equal to one.

We have provided the code skeleton including all the code and example mazes to get you started, which means you will only have to write the search functions. You should only modify search.py. **Use the provided API functions (e.g. getNeighbors) and do not modify code in files other than search.py.** Otherwise, the grader may be unable to run your code and/or may decide that your outputs are incorrect.

maze.py

- getStart() :- Returns a tuple of the starting position, (row, col)
- getObjectives() :- Returns a list of tuples that correspond to the dot positions, [(row1, col1), (row2, col2)]
- isValidMove(row, col) :- Returns the boolean **True** if the (row, col) position is valid. Returns **False** otherwise.
- getNeighbors(row, col) :- Given a position, returns the list of tuples that correspond to valid neighbor positions. This will return at most 4 neighbors, but may return less.

search.py

You can (and should) create additional test mazes to make sure your code is working properly and/or help you debug problems. The final evaluation mazes will be different from the shown examples.

Part 1: Finding a single dot

Find the shortest path from a given start state to a goal state. Implement the following search strategies:

- Breadth-first search (BFS)
- Depth-first search (DFS)
- Uniform-cost search (UCS)
- A* search

For further usage, you are suggested to directly implement the state representation, transition model, and goal test needed for **solving the problem in the general case of multiple dots**.

Observe the returned path length and state explored for each search strategy.

Part 2: Finding all corners

Now we consider a little harder search problem to experience the real power of A*. In this part, we define one type of objective called corner mazes. In corner mazes, there are only four dots, one in each corner. Our new search problem is to find the shortest path through the maze, starting from P and touching all four corners.

Grading will be based on total path length, computed within 4 seconds per maze.

Submissions

- ***<your metric number>_search.py*** - Your search.py python code
- ***<your metric number>_search.pdf*** - Short report describing the programming assignment and your solution.