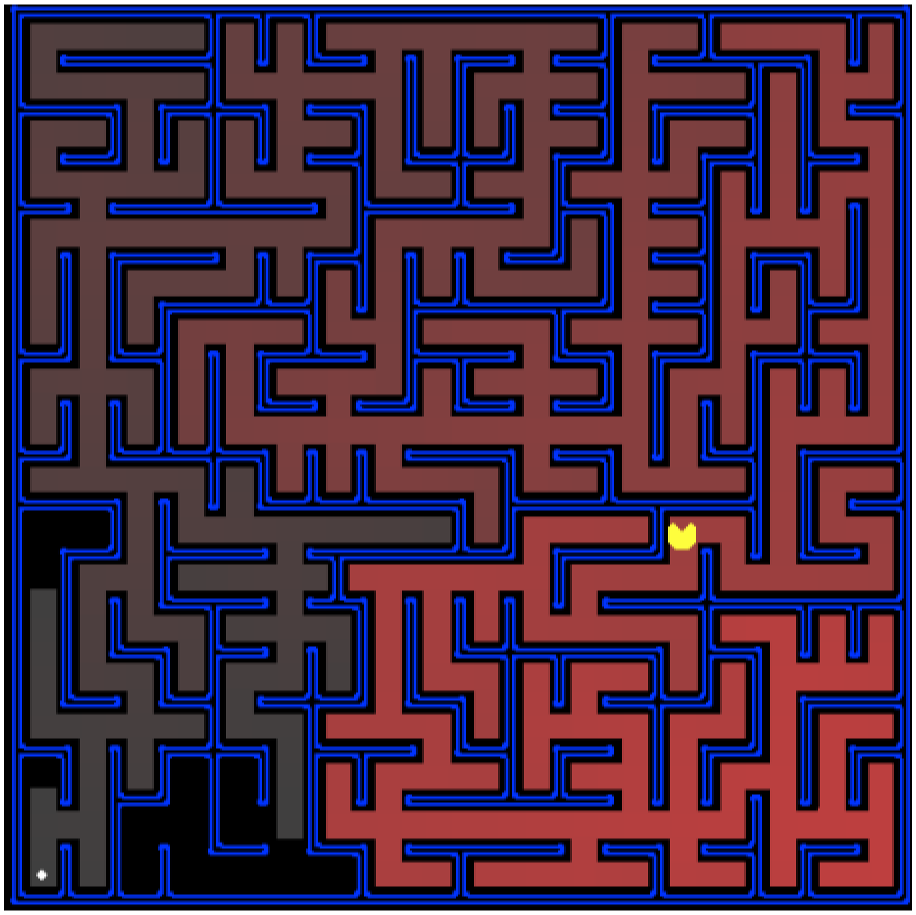
Search in Pacman



All those colored walls,
Mazes give Pacman the blues,
so teach him to search.

Project designed by: John DeNero & Dan Klein and modified by Luke Zettlemoyer

In this project, your Pac-Man agent will find paths through his maze world, both to reach a particular location and to collect food efficiently. You will build general search algorithms and apply them to Pac-Man scenarios.

The code for this project consists of several Python files, some of which you will need to read and understand in order to complete the assignment, and some of which you can ignore. You can download all the code and supporting files (including this description) as a zip archive.

Files you'll edit:

search.py	where all your search programs will reside.
searchAgents.py	where all your search-based agents will reside.

Files you don't need to understand but might want to look at:

pacman.py
The main file that runs Pac-Man games. This file describes a Pac-Man GameState type, which you use in this project.

game.py
The logic behind how the Pac-Man world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.

util.py
Useful data structures for implementing search algorithms.

Files you don't need to understand or look at:

graphicsDisplay.py	Graphics for Pacman
graphicsUtils.py	Support for Pacman graphics
textDisplay.py	ASCII graphics for Pacman
ghostAgents.py	Agents to control ghosts
keyboardAgents.py	Keyboard interfaces to control Pacman
layout.py	Code for reading layout files and storing their contents

What to submit: You will fill in portions of `search.py` and `searchAgents.py` during the assignment. You should submit these two files (only).

After downloading the code (`search.zip`), unzipping it and changing to the search directory, you should be able to play a game of Pac-Man by typing the following at the command line:

```
python pacman.py
```

Pac-Man lives in a shiny blue world of twisting corridors and tasty round treats. Navigating this world efficiently will be Pac-Man's first step in mastering his domain.

The simplest agent in `searchAgents.py` is called the GoWestAgent, which always goes West. This agent can occasionally win:

```
python pacman.py layout testMaze pacman GoWestAgent
```

But, things get ugly for this agent when turning is required:

```
python pacman.py layout tinyMaze pacman GoWestAgent
```

If pacman gets stuck, you can exit the game by typing `CTRL-C` into your terminal. Soon, your agent will solve not only tinyMaze, but any maze you want. o_O

Note that `pacman.py` supports a number of options that can each be expressed in a long way (e.g., `--layout`) or a short way (e.g., `-l`). You can see the list of all options and their default values via:

```
python pacman.py h
```

Also, all of the commands that appear in this project also appear in `commands.txt`, for easy copying and pasting. In UNIX/Mac OS X, you can even run all these commands in order with bash `commands.txt`.

In searchAgents.py, you'll find a fully implemented SearchAgent, which plans out a path through Pac-Man's world and then executes that path step-by-step. The search algorithms for formulating a plan are not implemented -- that's your job. First, test that the SearchAgent is working correctly by running:

```
python pacman.py l tinyMaze p SearchAgent a fn=tinyMazeSearch
```

The command above tells the SearchAgent to use tinyMazeSearch as its search algorithm, which is implemented in search.py. Pac-Man should navigate the maze successfully. Now it's time to write full-fledged generic search functions to help Pac-Man plan routes!

Depth-First Search

Implement the depth-first search (DFS) algorithm in the `depthFirstSearch` function in `search.py`. Your code should quickly find a solution for:

```
python pacman.py -l tinyMaze -p SearchAgent
python pacman.py -l mediumMaze -p SearchAgent
python pacman.py -l bigMaze -z .5 -p SearchAgent
```

The Pac-Man board will show an overlay of the states explored, and the order in which they were explored (brighter red means earlier exploration).

Breadth-First Search

Implement the breadth-first search (BFS) algorithm in the `breadthFirstSearch` function in `search.py`. Test your code the same way you did for depth-first search.

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
```