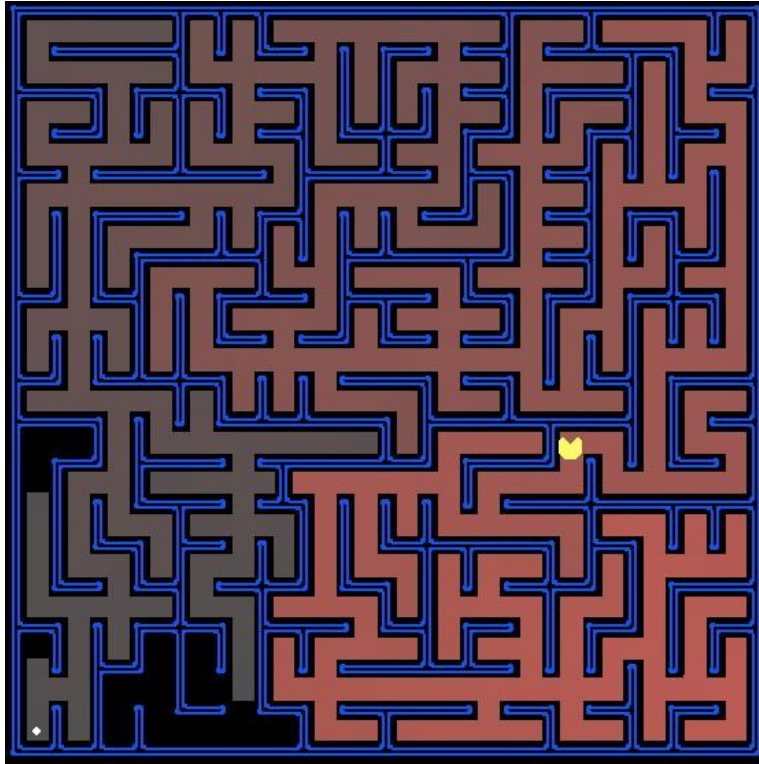# IT159IU: Artificial Intelligence
Lab#2/Assignment#2
Uninformed Search in Pac-Man

There are two exercises in this lab:

1. Depth-First Search (DFS)
2. Breadth-First Search (BFS)
3. Uniform-Cost Search (UCS)

Introduction

In this assignment, you will continue to work with the archive in the first lab assignment to help your Pac-Man agent find paths through his maze world to reach a particular location. You will build general search algorithms and apply them to many different Pac-Man scenarios.

Files you'll edit:

search.py   Where all your search algorithms will reside.

searchAgents.py Where all your search-based agents will reside.

Files you should look at but NOT edit:

`util.py`	Useful data structures for implementing search algorithms.
`pacman.py`	The main file that runs Pac-Man games. This file describes a PacMan `GameState` type, which you use in this lab.
`game.py`	The logic behind how the Pac-Man world works. This file describes several supporting types like `AgentState`, `Agent`, `Direction`, and `Grid`.

Finding a fixed food dot using Uninformed Search

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. As you work through the following questions, you might need to refer to the glossary of objects in the code (at the end of this handout).

First, test that the `SearchAgent` is working correctly by running:

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

```
PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l tinyMaze -p SearchAgent -a fn=tinyMazeSearch
[SearchAgent] using function tinyMazeSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 0
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

The command above tells the `SearchAgent` to use `tinyMazeSearch` as its search algorithm, which is implemented in `search.py`. This simply follows a fixed sequence of actions to demonstrate how the code works. Pac-Man should navigate the maze successfully.

Now it's time to write the generic search functions to help Pac-Man plan routes! Pseudocode for the depth-first search, breadth-first search and uniform-cost search algorithms might be useful for the implementation is shown below.

```
function UninformedSearch(problem) returns a list of actions
    initialize the frontier using the initial state of the problem
    initialize exploredSet to empty
    #For explored, use Pacman position as the key with a value True
    #initialize a set of states already explored
    while frontier is not empty do
        choose a leaf node and remove it from the frontier
        if the node contains a goal state
```

```
        return list of actions from start state to goal state
    add the state to the exploredSet
    for each successor of the node state
        if the successor state is not in exploredSet
            add node of the successor onto the frontier
return an empty list (i.e. no solution!)
```

Important note*:* All your search functions need to return a list of *actions* that will lead the agent from the start to the goal. These actions must be legal moves (valid directions, no moving through walls).

Hint: Algorithms for DFS, BFS and UCS differ only in the details of how the frontier is managed. So, concentrate on getting DFS right and then BFS should be relatively straightforward. Indeed, one possible implementation requires only a single generic search method which is configured with an algorithm-specific queuing strategy. Your implementation need *not* be of this form to receive full credit.

Hint: Make sure to check out the `Stack`, `Queue`, and `PriorityQueue` types provided to you in `util.py`.

EXERCISE 1: Implement the depth-first search algorithm in the `depthFirstSearch` function in `search.py`. Although DFS and BFS ignore the costs, you'll need them for later search methods. Your code should quickly find a solution for:

`python pacman.py -l tinyMaze -p SearchAgent –a fn=dfs`

```
PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l tinyMaze -p SearchAgent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 10 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 500
Average Score: 500.0
Scores:        500.0
Win Rate:      1/1 (1.00)
Record:        Win
```

`python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs`

```
PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l mediumMaze -p SearchAgent -a fn=dfs
[SearchAgent] using function dfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 146
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
```

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

```
 PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l bigMaze -p SearchAgent -a fn=dfs
 [SearchAgent] using function dfs
 [SearchAgent] using problem type PositionSearchProblem
 Path found with total cost of 210 in 0.0 seconds
 Search nodes expanded: 390
 Pacman emerges victorious! Score: 300
 Average Score: 300.0
 Scores:        300.0
 Win Rate:      1/1 (1.00)
 Record:        Win
```

The Pac-Man board will show an overlay of color for the states explored and the order in which they were explored (brighter red means earlier exploration). Is the exploration order what you would have expected? Does Pac-Man actually go to all the explored squares on his way to the goal?

Hint: The solution found by your DFS algorithm for mediumMaze should have a length of 130 (provided you push successors onto the frontier in the order provided by getSuccessors(); you might get 244 if you push them in the reverse order). Is this a least cost solution? If not, think about what depth-first search is doing wrong.

EXERCISE 2: Implement the breadth-first search algorithm in the breadthFirstSearch function in search.py. Use the same algorithm as shown in the above pseudocode. Test your code the same way you did for depth-first search.

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

```
 PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l tinyMaze -p SearchAgent -a fn=bfs
 [SearchAgent] using function bfs
 [SearchAgent] using problem type PositionSearchProblem
 Path found with total cost of 8 in 0.0 seconds
 Search nodes expanded: 15
 Pacman emerges victorious! Score: 502
 Average Score: 502.0
 Scores:        502.0
 Win Rate:      1/1 (1.00)
 Record:        Win
```

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

```
 PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
 [SearchAgent] using function bfs
 [SearchAgent] using problem type PositionSearchProblem
 Path found with total cost of 68 in 0.0 seconds
 Search nodes expanded: 269
 Pacman emerges victorious! Score: 442
 Average Score: 442.0
 Scores:        442.0
 Win Rate:      1/1 (1.00)
 Record:        Win
```

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

```
PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l bigMaze -p SearchAgent -z .5 -a fn=bfs --frameTime
0
[SearchAgent] using function bfs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.1 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Does BFS find a least cost solution?

Hint: If Pac-Man moves too slowly for you, try the option `--frameTime 0`.

EXERCISE 3: Implement the uniform-cost search algorithm in the `uniformCostSearch` function in `search.py`. Does UCS find a least cost solution? How many nodes are expanded? You should now observe successful behavior in all three of the following layouts, where the agents below are all UCS agents that differ only in the cost function they use (the agents and cost functions are written for you):

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
```

```
PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
```

```
PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
Path found with total cost of 1 in 0.0 seconds
Search nodes expanded: 186
Pacman emerges victorious! Score: 646
Average Score: 646.0
Scores:        646.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

```
PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
Path found with total cost of 68719479864 in 0.0 seconds
Search nodes expanded: 108
Pacman emerges victorious! Score: 418
Average Score: 418.0
Scores:        418.0
Win Rate:      1/1 (1.00)
Record:        Win
```

*Alternative:

```
PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l tinyMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 8 in 0.0 seconds
Search nodes expanded: 15
Pacman emerges victorious! Score: 502
Average Score: 502.0
Scores:        502.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 68 in 0.0 seconds
Search nodes expanded: 269
Pacman emerges victorious! Score: 442
Average Score: 442.0
Scores:        442.0
Win Rate:      1/1 (1.00)
Record:        Win
```

```
PS E:\Homework\AI\AIEXT - edited\search\template> python pacman.py -l bigMaze -p SearchAgent -z .5 -a fn=ucs --frameTime
 0
[SearchAgent] using function ucs
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 210 in 0.0 seconds
Search nodes expanded: 620
Pacman emerges victorious! Score: 300
Average Score: 300.0
Scores:        300.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Note: You should get very low and very high path costs for the StayEastSearchAgent and StayWestSearchAgent respectively, due to their exponential cost functions (see searchAgents.py for details).

Object Glossary

Here's a glossary of the key objects in the code base related to search problems, for your reference:

`SearchProblem (search.py)`

> A `SearchProblem` is an abstract object that represents the state space, successor function, costs, and goal state of a problem. You will interact with any `SearchProblem` only through the methods defined at the top of `search.py.`

`PositionSearchProblem (searchAgents.py)`

> A specific type of `SearchProblem` that you will be working with --- it corresponds to searching for a single pellet in a maze.

```
Search Function
```
A search function is a function which takes an instance of `SearchProblem` as a parameter, runs some algorithm, and returns a sequence of actions that lead to a goal. Example of search functions are `depthFirstSearch` and `breadthFirstSearch`, which you have to write. You are provided `tinyMazeSearch` which is a very bad search function that only works correctly on `tinyMaze.`

```
SearchAgent
```
`SearchAgent` is a class which implements an Agent (an object that interacts with the world) and does its planning through a search function. The `SearchAgent` first uses the search function provided to make a plan of actions to take to reach the goal state, and then executes the actions one at a time.

What to submit

1. Fill out the table below:

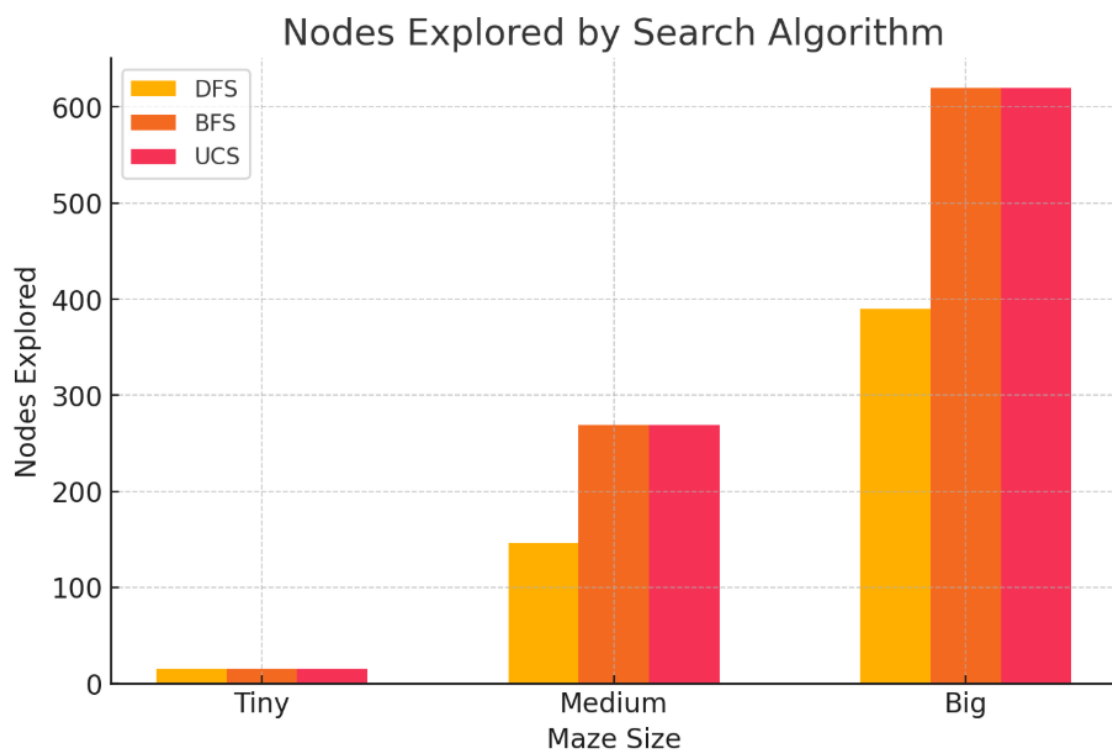| Maze	Depth-First Search			Breadth-First Search			Uniform-Cost Search		
	#nodes explored	Solution length	Is it optimal?	#nodes explored	Solution length	Is it optimal?	#nodes explored	Solution length	Is it optimal?
tiny	15	10	No	15	8	Yes	15	8	Yes
medium	146	130	No	269	68	Yes	269	68	Yes
big	390	210	No	620	210	Yes	620	210	Yes

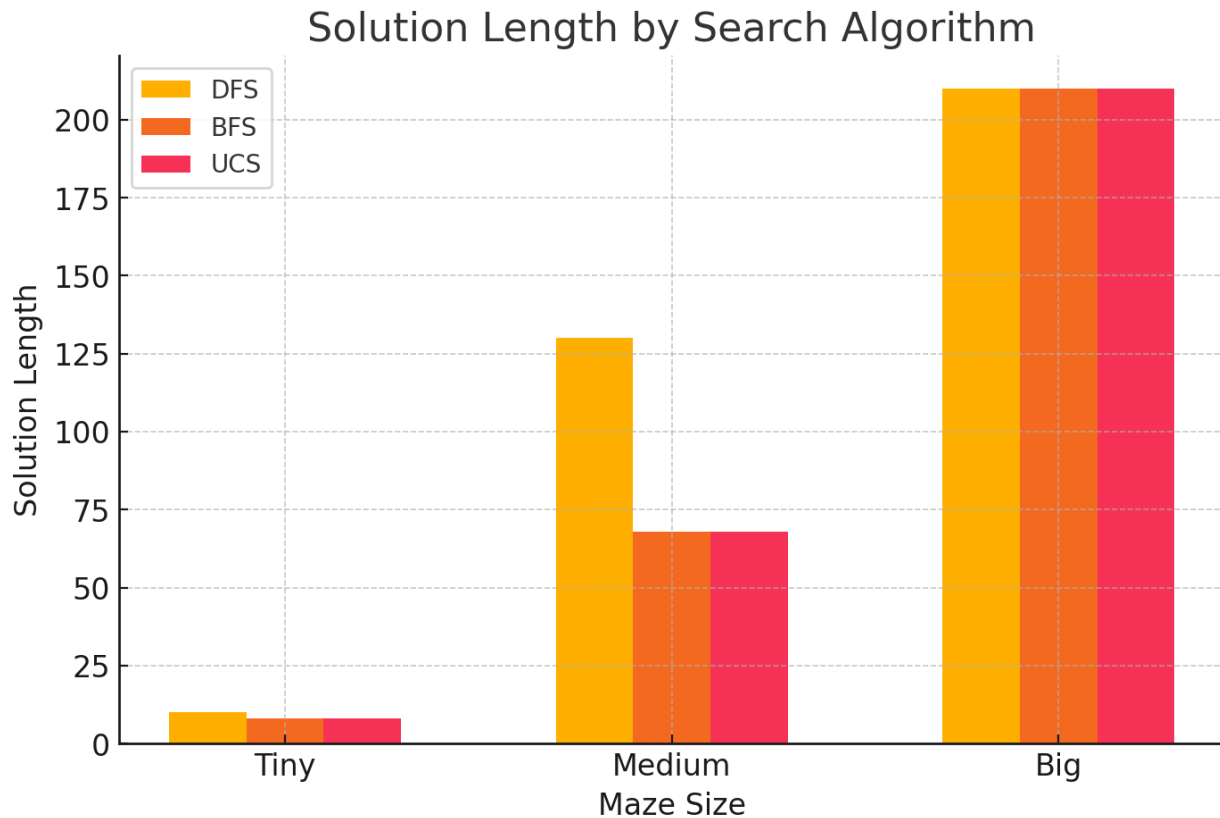Figure 1. Node Explored by Search Algorithm

Figure 2. Solution Length by Search Algorithm

2. Based on the above, a short discussion/reflection of how the searches compare.

- Depth-First Search (DFS)
    - DFS explores fewer nodes in small mazes but performs poorly in larger mazes, leading to longer solution paths.
    - It does not guarantee an optimal solution, as it prioritizes depth over efficiency.
- Breadth-First Search (BFS)
    - BFS explores more nodes than DFS but guarantees an optimal solution in terms of the shortest path.
    - As maze size increases, BFS requires significantly more nodes to be explored, making it computationally expensive.
- Uniform-Cost Search (UCS)
    - UCS behaves similarly to BFS in this scenario since all moves have equal cost, always finding the optimal solution.
    - It explores the same number of nodes as BFS and provides the shortest path.