CS-361L Artificial Intelligence Lab 04

Type of Lab: Open Ended **Weightage: 5%**

CLO 1: Apply Informed and **Uninformed** Search Techniques and build the ability to theoretical and practical understanding of Blind and Informed machine search and machine learning techniques.

Student implements Uninformed (**BFS and UCS**) techniques to search goals for Pacman	**Cognitive/Understanding**	CLO1	Rubric A
Demonstrate ethical and professional responsibilities involved in completion of Tasks	**Affective/Valuing**	(CLO6)	Rubric B

Reference: The Lab contents are extracted from the BerkeleyX/CS188

Rubric A: Cognitive Domain

Evaluation Method: GA shall evaluate the students for Question 1-3 according to following rubrics.

CLO	**0**	**1**	**2**	**3**	**4**
CLO1	Student was not able to complete challenge 01 (BFS)	Student was not able to complete challenge 02 (BFS vs DFS)	Student found the data structure for UCS implementation from util and understood how to implement UCS	Student tried to implement UCS, but the code had some errors but logically the try was sensible	The Pacman is navigating and reaching the goal with best utility by eating all pallets
Roll Number					

Reference: The Lab contents are extracted from the BerkeleyX/CS188

Rubric B: Affective Domain: Lab Staff shall help GA in evaluation of the students for their CLO 6

CLO 6	**0**	**1**	**2**	**3**
Demonstrate ethical and professional responsibilities involved in completion of Tasks.	Student was not on time in the lab	Student showed some unethical behavior and was late in lab	Student was on time and showed some unethical behavior	Student was obedient and showed ethical behavior
Roll Number				

Reference: The Lab contents are extracted from the BerkeleyX/CS188

Lab 04: Implementation of Breadth First Search (BFS) and its comparison with the Uniform Cost Search

Challenge 01: Breadth First Search

Implement the breadth-first search (BFS) algorithm in the `breadthFirstSearch` function in `search.py`. Again, write a graph search algorithm that avoids expanding any already visited states. Test your code the same way you did for depth-first search.

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

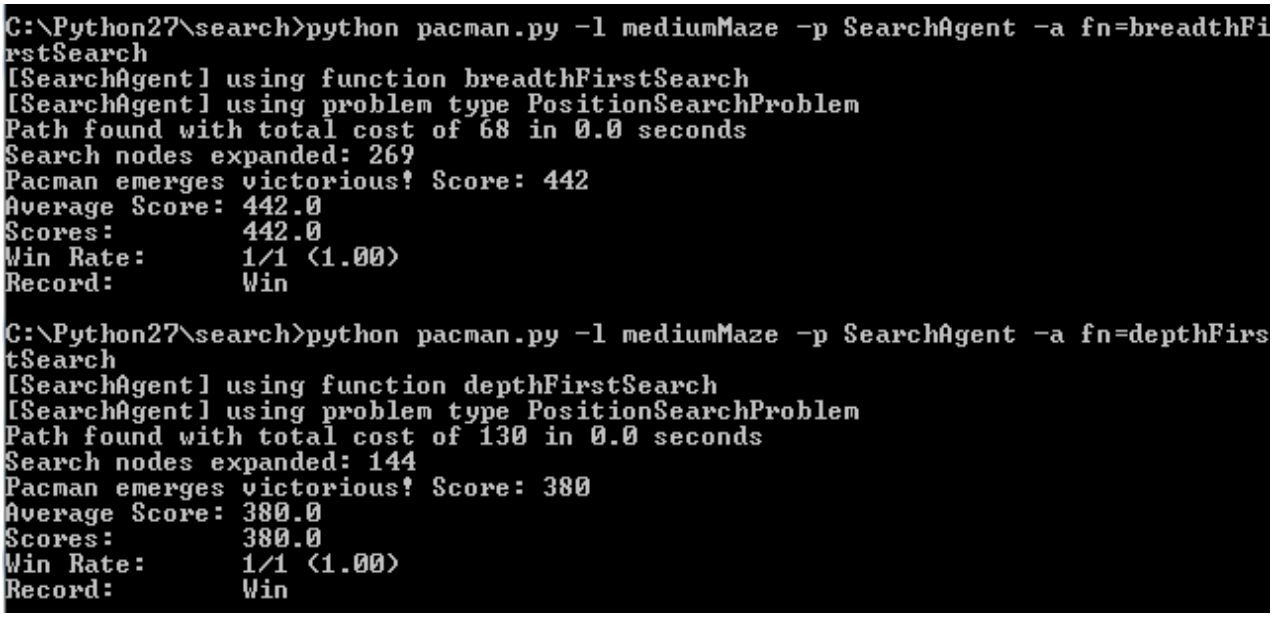
```
python pacman.py -l bigMaze -p SearchAgent -a fn=breadthFirstSearch -z .5
```

Does BFS find a least cost solution? If not, check your implementation.

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

Challenge 02 Comparison Between Breadth First Search and Depth First Search.

1. Run both algorithms one by one and check the output on the command prompt. You shall see different parameters such as total Cost, search node expanded and average score.



```
C:\Python27\search>python pacman.py -l mediumMaze -p SearchAgent -a fn=breadthFi
rstSearch
[SearchAgent] using function breadthFirstSearch
[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

C:\Python27\search>python pacman.py -l mediumMaze -p SearchAgent -a fn=depthFirs
tSearch
[SearchAgent] using function depthFirstSearch
[SearchAgent] using problem type PositionSearchProblem
Path found with total cost of 130 in 0.0 seconds
Search nodes expanded: 144
Pacman emerges victorious! Score: 380
Average Score: 380.0
Scores:        380.0
Win Rate:      1/1 (1.00)
Record:        Win
```

Reference: The Lab contents are extracted from the BerkeleyX/CS188

Challenge 03:

While BFS will find a fewest-actions path to the goal, we might want to find paths that are "best" in other senses. Consider `mediumDottedMaze` and `mediumScaryMaze`.

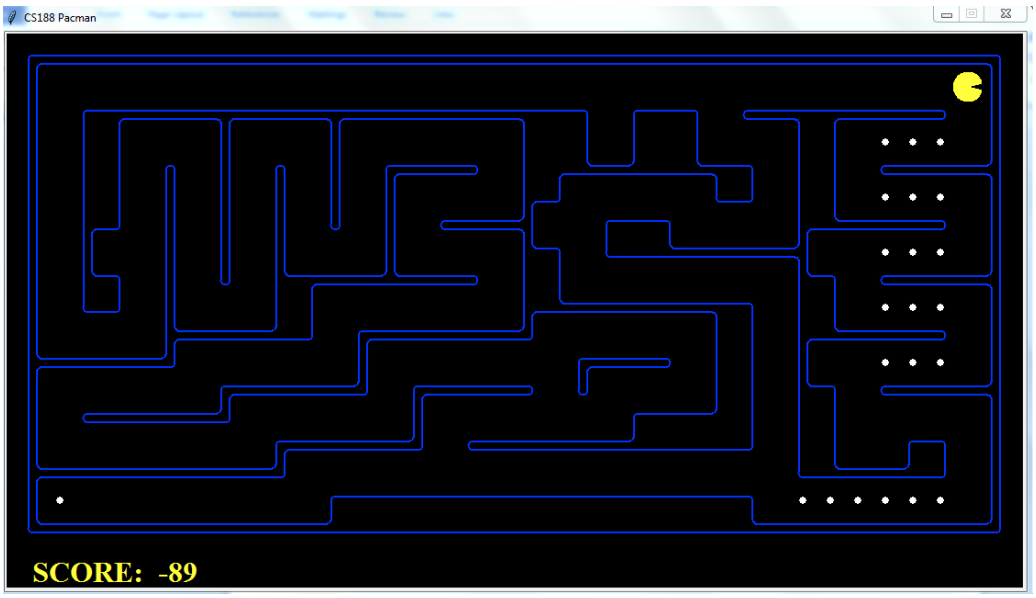
By changing the cost function, we can encourage Pacman to find different paths. For example, we can charge more for dangerous steps in ghost-ridden areas or less for steps in food-rich areas, and a rational Pacman agent should adjust its behavior in response.

Implement the uniform-cost graph search algorithm in the `uniformCostSearch` function in `search.py`. We encourage you to look through util.py for some data structures that may be useful in your implementation. 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. You can get the cost of each step from the triplet of state.

Hint: Print the state information and you shall see the cost of each state.

Step 1: To understand UCS, the program has built in mediumDottedMaze that maze return the utility according to the available pallet (food) on any state or not. You can view and paly around the maze with following command

python pacman.py -l mediumDottedMaze



Step 2: To understand how BFS work on the maze, you need to go Search.py file and edit the uniformCostFunction such that it should return the same value as the BFS is returning.

In other words, edit the function **def uniformCostSearch(problem):** and call breadthFirstFunction from this.

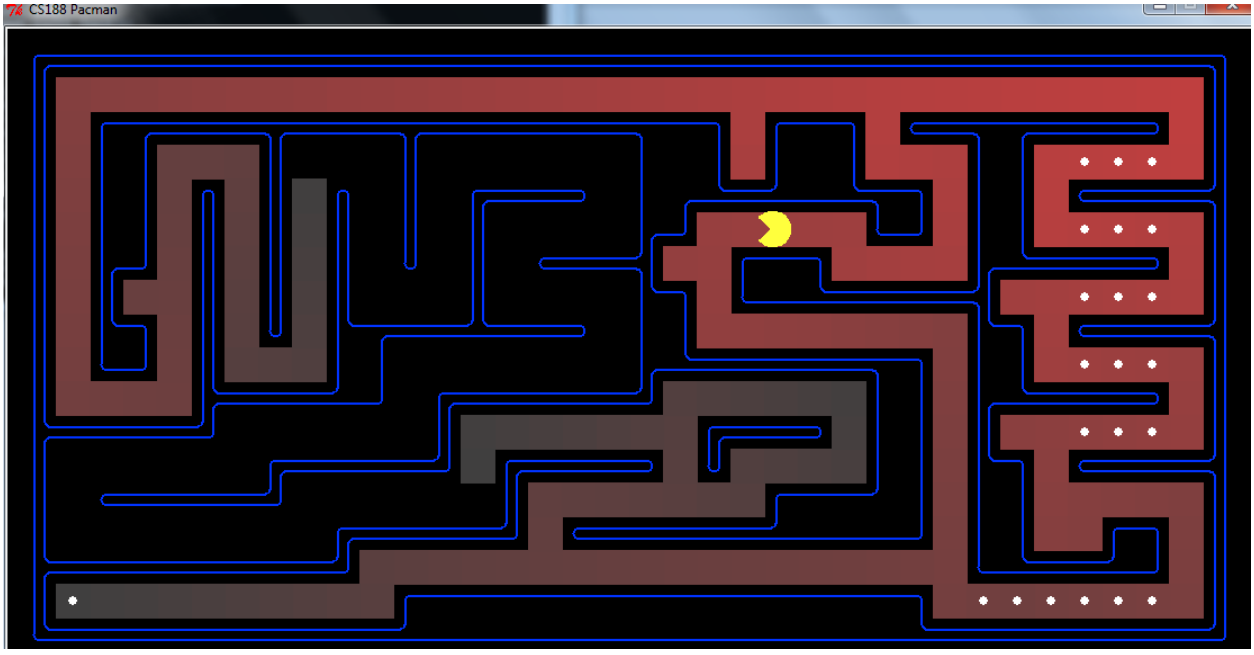
Reference: The Lab contents are extracted from the BerkeleyX/CS188

```
def uniformCostSearch(problem):
    return breadthFirstSearch(problem)
    #return uniformCostSearch_work(problem)
```

Now run following command to check the behavior of the searchAgent.

Python pacman.py –l mediumDottedMaze –p StayEastSearchAgent

Output of the command:

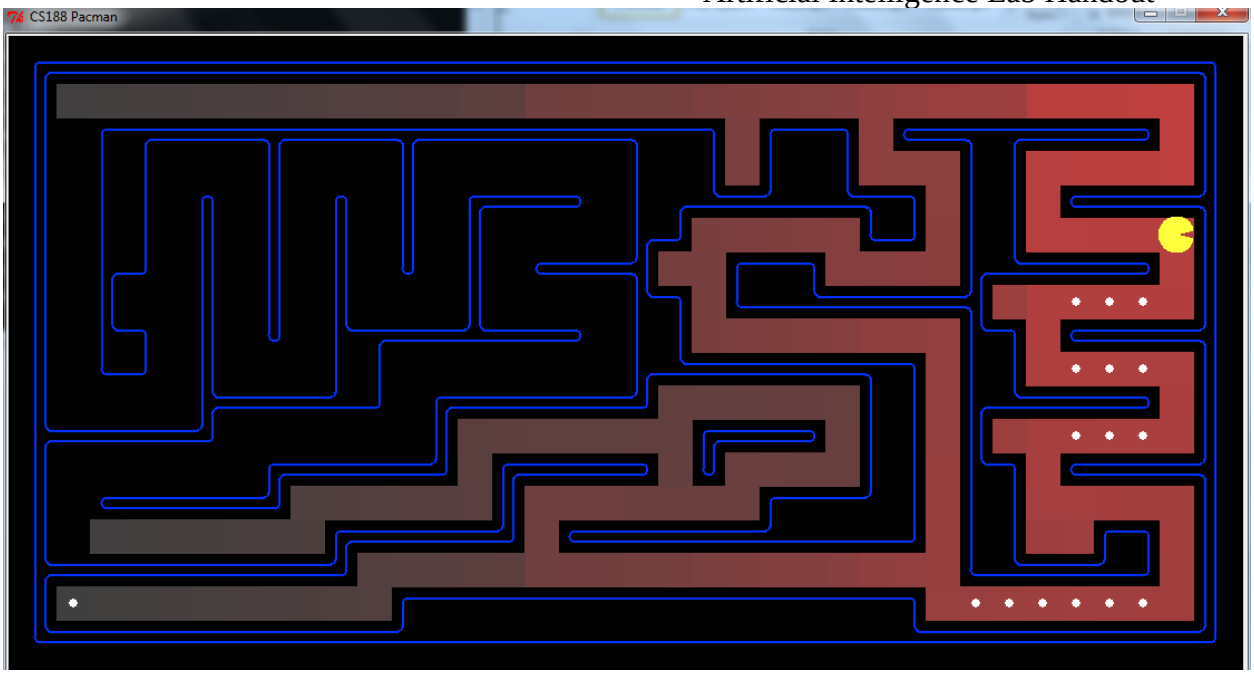


Note: Actually, in this step, we are using the breadth function as uniform cost search to understand that the optimality BFS is to find the nearest path, but one agent has some other optimality or utility. For example, for this specific Maze the utility of the search Agent is to eat as many as dots and not the shortest path. However, BFS is not considering that utility. Your job is to write down code such that the agent understands its utility function and eats all the food. So, the path of your solution should be like of following output.

Note: Now write down the program for UniformCostSearch function and run following command to view the output.

python pacman.py -l mediumDottedMaze -p StayEastSearchAgent

Reference: The Lab contents are extracted from the BerkeleyX/CS188



Challenge 04 (Take Home):

For dotted maze, run three commands separately

Python pacman.py –l mediumDottedMaze –p SearchAgent –a fn=dfs

Python pacman.py –l mediumDottedMaze –p SearchAgent –a fn=bfs

Python pacman.py –l mediumDottedMaze –p SearchAgent –a fn=ucs

1. **Fill following table with the information.**

	DFS	BFS	UCS
Total Cost			
Nodes Expanded			
Score			

2. **Why total cost is Greater in DFS and Less in BFS?**
3. **Why Nodes Expanded are Greater in BFS and Less in DFS?**
4. **Why the Score is Greater in BFS and Less in DFS?**
5. **Write a note which compares both algorithms in term of these parameters.**
6. **Compare BFS with UCS with reasoning.**

Reference: The Lab contents are extracted from the BerkeleyX/CS188