CS512– Artificial Intelligence

Instructor : Shashi Shekhar Jha (shashi@iitrpr.ac.in)

Lab Assignment - 1 | Due on 02/02/2020 2400 Hrs | 50 Marks

Submission Instructions:

All submission is through google classroom in one zip file. In case you face any trouble with the submission, please contact the TAs:

- Armaan Garg, 2019CSZ0002@iitrpr.ac.in
- Rahul Kumar Rai, 2018csz0004@iitrpr.ac.in

or you can post your queries in the classroom.

Your submission must be your original work. Do not indulge in any kind of plagiarism or copying. Abide by the honour and integrity code to do your assignment.

As mentioned in the class, late submissions will attract penalties.

Penalty Policy: There will be a penalty of 5% for every 24 Hr delay in the submission. E.g. for first 24 Hr delay the penalty will be 5%, for submission with a delay of >24 Hr and < 48 Hr, the penalty will be 10% and so on.

You submission must include:

- A legible PDF document with all your answers to the assignment problems, stating the reasoning and output.
- A folder named as 'code' containing the scripts for the assignment along with the other necessary files to run yourcode.
- A README file explaining how to execute your code.

Naming Convention:

Name the ZIP file submission as follows: ***Name_rollnumber_Assignmentnumber.zip***
E.g. if your name is ABC, roll number is 2017csx1234 and submission is for lab1 then you should name the zip file as: ABC_2017csx1234_lab1.zip

General Instructions

The purpose of this assignment is to exercise your basic understanding of intelligent agents, state space search, and to help you apply these concepts simulate the behaviour of search algorithms in *OpenAI Gym* environment.

"Gym is a toolkit for developing and comparing reinforcement learning algorithms. It supports teaching agents everything from walking to playing games like Pong or Pinball."

In this assignment, you will be experimenting with different AI search techniques that we discussed in class. You are provided with a starter code for this project. The code consists of several Python files, some of which you will need to read and understand to complete the assignment, and some of which you can ignore.

Instructions according to UBUNTU OS
NOTE: Work with Python3

Install Gym
Reference link for installation of gym environment is available at:
http://gym.openai.com/docs/

The template code is provided in the Assignment1.zip folder available with this assignment post at google classroom.

Download the Assignment1.zip and extract the contents.
Go to downloaded directory and run command:

```
pip3 install -e . --user
```

Save the world from Cersei Lannister

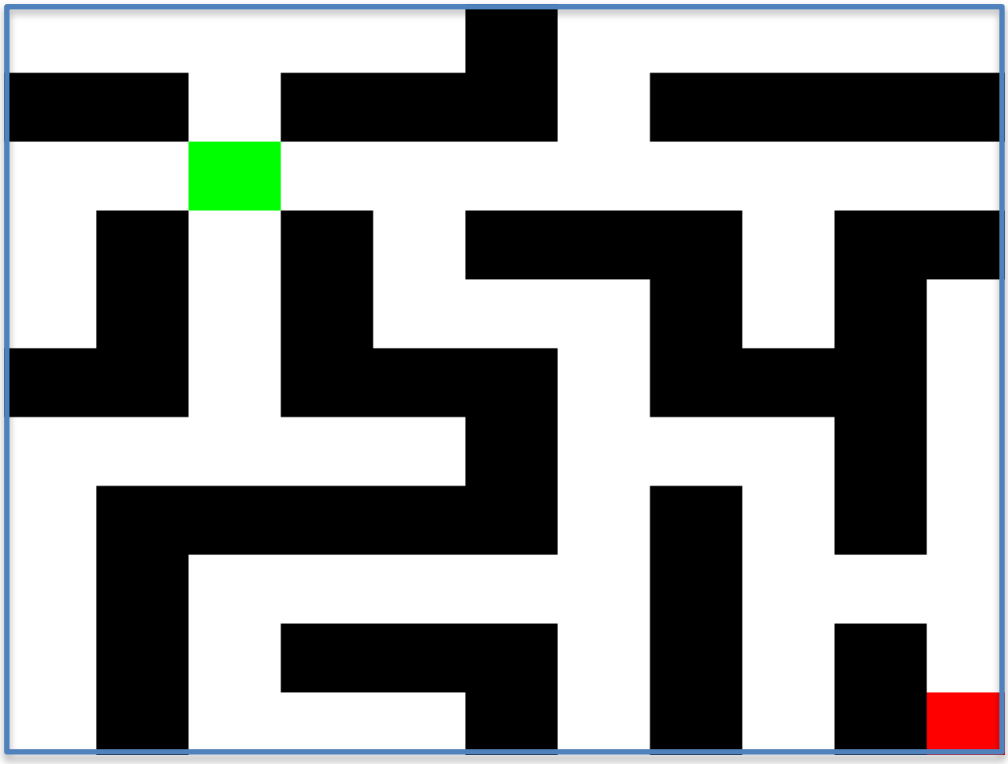
Melisandre, also known as red women was born in the city of Melony, and is known to have supernatural powers. She has been known to be lived for 2000 years and used her sorcery to save the world from evil Lannisters. It has been recently known that her immortality comes from the pendant she wears and few wise men say that the owner can only live for a few seconds without the magical pendant.

But on a very dark day, evil Cersei Lannister stole her pendant and hidden herself into an unknown city of labyrinths. Without the pendant, Melisandre will turn into ashes very soon and the world will be doomed with the evils of Lannisters. A warrior from the North, Ser Jon Snow has been put to the task of finding the pendant and bring back the balance to the world. Thanks to Jon Snow, the location of where Cercei is hiding is now known in the city of labyrinths. But the paths in this haunted city of labyrinths is so complicated that it will take ages for a person to reach to Cercei and recover the pendant. Ser Jon Snow is alone and needs reinforcements of the troops. The clock is ticking and Melisandre, the red women, would turn to ashes if the pendant is not recovered soon.

You have been approached as an expert of AI to find the path to Cercei's location in a very fast and efficient manner in the city of labyrinths. Now, the whole balance of the world lies on your understating of AI techniques. You plan to create 'Grand_Maester' an AI program which could tell the path to Cercei's location in mere seconds. Grand_Maester is an intelligent system that can work in any known/unknown environment and could perform various intelligent problem solving tasks.

If Grand_Maester gets stuck while executing, you can exit the search by typing CTRL-c into your terminal.

The figure below shows a snapshot of the city of labyrinth with Green block showing the location of the castle where troops are stationed and Red block showing the location of Cercei.



Files you'll edit:	
cair_maze/pathfinding.py	Where all of your search algorithms will reside.
Files you might want to look at:	
cair_maze/maze_game.py	The main file that runs Maze games. This file describes a Maze GameState type and Actions, which you use in this project.
cair_maze/maze.py	The logic behind how the Maze world is built. This file describes several supporting types like ActionSpace, StateSpace and Grid.

cair_maze/algorithms.py	Useful data structures for implementing search algorithms.
gym_maze/env/ maze_env.py	How environment is built in OpenAI Gym
cair_maze/mechanics.py	The mechanics behind the working and the structure of the game
Supporting files you can ignore:	
gym_maze/env/ no_maze_env.py	Basic environment for maze game
setup.py	The setup file
__init__.py	import file
documentation/ state_space.py	State Space layout file
Execution file:	
example/example_2.py	After writing down the search code you could try out the UI implementation by running this file. Correct implementation: You should see source pointer moving towards target based on search algorithm.

Files to Edit and Submit: You will fill in portion of `pathfinding.py` in this assignment. You should submit this file with your code and comments. Please *do not* change the other files in this distribution or submit any of our original files other than these files.

Evaluation:

Please *do not* change the names of any provided functions or classes within the code, or you will wreak havoc on the evaluation system. However, the correctness of your implementation -- will be the final judge of your score. We will review and grade assignments individually to ensure that you receive due credit for your work.

NOTE: In each question 2 Marks are for calculating the computing time and memory required by the search algorithms to find the solution. Further, 1 mark will be awarded for mentioning the number of nodes expanded by the search algorithms in order to find the final solution. Show and compare these parameters using graphs in a separate doc (this doc must be submitted as pdf). Your code must clearly show the implementation for capturing the above mentioned parameters while performing the search. For each algorithm you could write different search functions in the same file and call accordingly or create separate files.

Question 1 (5 points):
Finding Cercei using Depth First Search

In `maze_game.py`, you'll find a fully implemented SearchAgent, which plans out a path through labyrinth and then executes that path step-by-step. The search algorithms for formulating a plan are not implemented -- that's your job.

It's time to write full-fledged generic search functions to help Grand_Maester plan routes. Remember that a search node must contain not only a state but also the information necessary to reconstruct the path (plan) which gets to that state.

Important note: All of your search functions need to return a list of *actions* that will lead the troops from the start to the goal. These actions all have to be legal moves (valid directions, no moving through walls).
Helping note: Make sure to **use** the Stack, Queue and PriorityQueue data structures. These data structure implementations have particular properties.

Hint: Each algorithm is very similar. Algorithms for DFS, BFS, UCS and A* differ only in the details of how the frontier is managed. So, concentrate on getting DFS right and the rest should be relatively straightforward. Indeed, one possible implementation requires only a single generic search method which is configured with an algorithm- specific queuing strategy.

Implement the depth-first search (DFS) algorithm in the search function in `pathfinding.py`. To make your algorithm *complete*, ***write the graph search version of DFS***, which avoids expanding any already visited states.

To test the code run `example_2.py`.
Each step incurs a cost 2 units, do find the cost in case of DFS.
Does DFS find a Solution?

Question 2 (5 points): Breadth First Search

Implement the breadth-first search (BFS) algorithm in the search function in `pathfinding.py`. To make your algorithm *complete*, **write the graph search version of BFS**, which avoids expanding any already visited states.

To test the code run `example_2.py`

Does BFS find a solution? If not, check your implementation.
Each step incur 2 units cost, mention the cost of search using your BFS implementation.

Question 3 (5 points): Varying the Cost Function

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

Use the following cost function:
Vertical Movement towards the target incur 1 unit cost; otherwise the cost is 2 units.
Implement the uniform-cost graph search algorithm in `pathfinding.py`

Do find the cost in case of UCS as well.

Question 4 (10 points): A* search

Implement A* graph search in the search function in `pathfinding.py`. A* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information).

You can test your A* implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic or Diagonal distance heuristic.

Cost function: (same as Question 3)
Do find the cost in case of A*.

Question 5 (10 points): Working with new environment

Use your implementation of A* search in a new OpenAI Gym Environment of your choice and show how efficiently you can score in that environment. The real power of A* search will be more apparent with a multi-goal search problem. Remember that A* can work with fully observable environments. Following link provides the list of environments available in OpenAI Gym: https://gym.openai.com/envs/#classic_control

Best of Luck!!!