CSE 571: Artificial Intelligence
Homework #1

Due Date: February 7^{th}, 2018, 11:59 PM

Automated plagiarism detection software will be used on the code as well as text portions of the assignment.

Submission Instructions

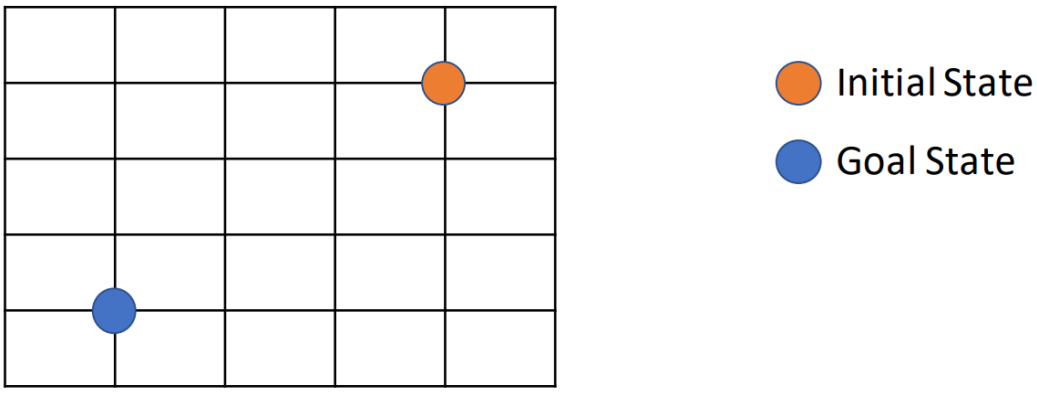
Please make sure that you submit your homework in a specific way.

- Answers for all the question except Question 3 should be in a pdf named as ⟨your_last_name⟩_⟨your_first_name⟩_⟨your_ASUID⟩.pdf. This pdf should include the plots for Question 8.
- Answer for Question 3 should be in a txt file named as ⟨your_last_name⟩_⟨your_first_name⟩_⟨your_ASUID⟩_q3.txt
- For Question 8's code, submit your ***search_ algorithm.py*** file names as "search_algorithm_⟨your_last_name⟩_⟨your_first_name⟩_⟨your_ASUID⟩.py".

DO NOT ZIP THE FILES.

Question 1 [10 points]

Consider the environment given below. The agent can move only on horizontal or vertical edges. Diagonal movements are not possible. Prove that Euclidean and Manhattan distances are admissible heuristics. If diagonal movements were allowed, would both heuristics still be admissible? Justify your answer.



Question 2 [10 points]

Invent a heuristic function for the 8-puzzle that sometimes overestimates, and show how it can lead to a sub-optimal solution on a particular problem. Prove that if h never overestimates by more than c, A^* using h returns a solution whose cost exceeds that of the optimal solution by no more than c. Which nodes will A^* with h expand definitely?

Question 3 [10 points]

Find the graphs with your name from **Files->Homework1->Graphs**. Use that graph to perform **BFS, DFS, UCS, GBFS and** A^*. Find the file with you name and download that. Now perform the following operations on the downloaded graph.
(a) Perform BFS and return the path as well as path cost from Start to first expanded Goal node.
(b) Perform DFS and return the path as well as path cost from Start to first expanded Goal node.
(c) Perform UCS and return the path as well as path cost from Start to first expanded Goal node.
(d) Perform GBFS and return the path as well as path cost from Start to first expanded Goal node.
(e) Perform A* and return the path as well as path cost from Start to first expanded Goal node.

All ties should be broken alphabetically.

Submit your answers for this question in a .txt file strictly in the given format:
BFS,{pathCost},{⟨ nodes of your paths seperated by comma⟩}
DFS,{pathCost},{⟨ nodes of your paths seperated by comma⟩}
UCS,{pathCost},{⟨ nodes of your paths seperated by comma⟩}
GBFS,{pathCost},{⟨ nodes of your paths seperated by comma⟩}
A*,{pathCost},{⟨ nodes of your paths seperated by comma⟩}
For example, if your path for BFS is Start, A, C, Goal and path cost is 15, then your txt file should include:
BFS,15,Start,A,C,Goal
Similarly for DFS, UCS and A*. All your answers for this question should be in one txt file named as ⟨your_last_name⟩_⟨your_first_name⟩_⟨your_ASUID⟩_q3.txt

Question 4 [15 points]

Formulate the following problems as search problems. Choose a formulation that is precise enough to be implemented.

1. Using only four colors, you have to color a planar map in such a way that no two adjacent regions have the same color.
2. A 3 feet tall monkey is in a room where some bananas are suspended from 8 feet ceiling. He would like to get the bananas.The room contains two stackable, movable, climbable 3 feet high crates.
3. You have three jugs, measuring 12 gallons, 8 gallons and 3 gallons, and a water faucet. You can fill the jugs up or empty them from one to another or onto the ground. You need to measure out exactly one gallon.

Question 5 [5 points]

Give an example state space where Iterative Deepening will perform worse than Depth First Search in terms of number of nodes expanded.

Question 6 [5 points]

Describe the procedure for converting an admissible but not consistent heuristic to a consistent heuristic.

Question 7 [5 points]

Describe the procedure for converting a goal based agent to a utility based agent.

Question 8 [15 points]

Implement BFS, UCS, GBFS, and A* using the provided ROS/Gazebo platform (see Appendix A). Evaluate the performance of these algorithms for the robot navigation problem where the Turtlebot3 robot needs to hit the target object (and only the target object).

For the evaluation, use the following experimental settings with the provided problem generator:

Dimension of the square grid	Number of blocked edges
2	5
4	16
8	58
16	218

Figure 1: Settings for Question 8 (left) and an example of a generated 2x2 grid without any blocked edges (right). Your robot will travel along the edges of this grid. Grid dimension denotes the number of segments per edge of a square grid. The distance between two neighboring points on the same edge is 0.5m. Thus, the grid with dimension 2 occupies a 1m x 1m square.

1. On a single graph, plot the time taken to search for a path to the goal by each of the following algorithms: BFS, UCS, GBFS, A*. Use the Manhattan-distance heuristic (h_M) for GBFS and A*. X-axis should represent the grid dimension, Y-axis the time taken as an average of 5 runs per grid dimension, and different data series should correspond to the 4 different algorithms.**[10 points]**
2. Construct a different heuristic (say h_s) for this problem. Describe h_s as a function mapping states to real numbers. Repeat the plotting exercise above with two variants each of GBFS and A*, one using h_M and the other using h_s. Extra credit if GBFS or A* using your h_s outperform the corresponding algorithm with h_M!**[5 points + 5 bonus]**
3. Place all your search algorithms inside the ***/search/scripts/search_ algorithm.py***. You can find it in the ROS package that you downloaded for this assignment.
4. Rename the file as "search_algorithm_⟨your_last_name⟩_⟨your_first_name⟩_⟨your_ASUID⟩.py" and submit it.

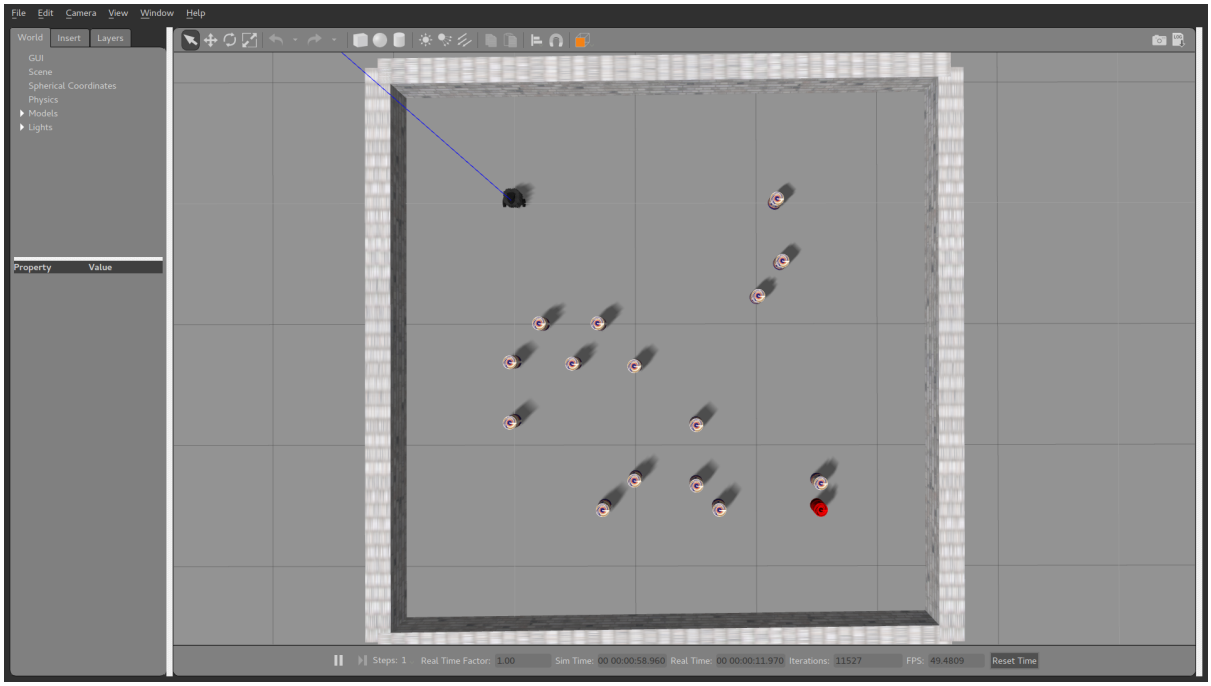


Figure 2: Example Environment

A ROS/Gazebo Platform for Testing Search Algorithms

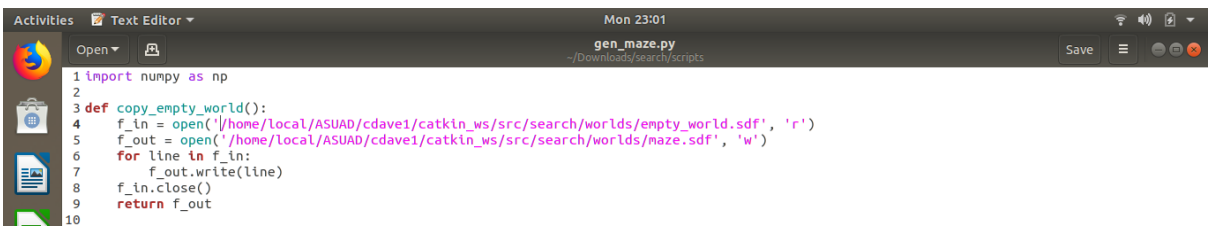
A.1 Problem Setup

In this implementation, we will build on the Turtlebot3 resources you installed in the previous homework. The objective is to get the Turtlebot3 to navigate a cluttered environment Fig 2. The state of the turtlebot will be represented as x, y, ϕ where x and y are coordinates ϕ is an orientation in $\{NORTH, SOUTH, EAST, WEST\}$. The set of possible actions includes $\{MoveF, MoveB, TurnCW, TurnCCW\}$. *Move* actions move the turtlebot forward or backward along an edge, and *Turn* actions rotate it clockwise or counterclockwise.

A.2 Required Installation and Configuration

Please follow the below steps for downloading and configuring the required ROS package for Q8.

- Download the ROS package named *search.zip* from the canvas course page. Extract it and put folder ***search*** inside the *src/* folder of your catkin workspace. Change permission of all the python files in scripts folder to allow execution.
- Go to the *worlds/* folder and copy the complete path of the file *empty_ world.sdf* (right click on the file, select properties and there you will get the complete path). Now, open the file *gen_ maze.py*, and paste the path in line no.4 and use the same path for the file *maze.sdf* at line no.5. **Please follow this step very carefully or else you won't be able to view your maze in gazebo**.



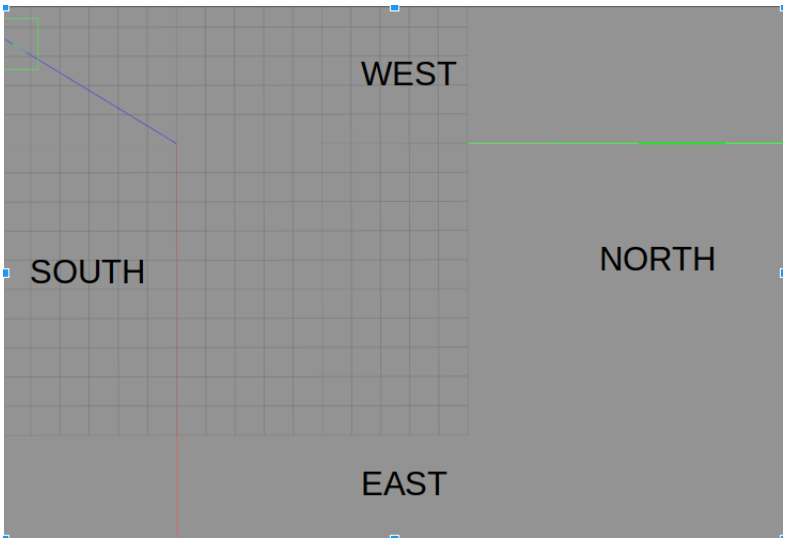
- From your **catkin_ws** folder, run the command ***catkin_ make*** to build the files.
- From your **catkin_ws** folder, run the command ***source devel/setup.bash*** to source the files that you build in the previous step.
- Make sure that you have all the Turtlebot3 packages installed that were required in the last assignment inside the src folder of your catkin workspace.

A.3 Helper Functions That You Need to Use

The *search.zip* package provides a number of helper functions that you will need to implement your search algorithms. All the helper functions are provided under the file named *problem.py* inside the *scripts/* folder and so we would suggest you to go through it once.

Please go through this information very carefully. If you have any doubts, clarify them as soon as possible.

- The function ***get_ initial_ state()*** takes no argument as input and returns the initial state of the Turtlebot3 in the form of ***State*** class, which has three members (x-cordinate, y-cordinate, orientation).
- The function ***is_ goal_ state()*** takes *State* of the Turtlebot3 as argument and returns 1 if it is the goal state or 0 if it is not the goal state.
- The function ***get_ successor()*** takes an *State* and the action being performed by the Turtlebot3 as arguments and returns the successor state and cost of the action.
- The function ***exec_ action_ list*** takes a plan as input and attempts to execute it in the Gazebo simulator. Make sure to use this function to execute your plan at the end of search. Our grading script will use the topic that this function publishes to, in order to evaluate your algorithm.
- The ***search/scirpt/random_ walk.py*** codes a random-walk algorithm that you can inspect and run to see how the system works.
- The template file */search/scripts/search_ algorithm.py* provides a template that you should build upon to implement search algorithms.



A.4 Testing and Evaluating Custom Search Algorithms

With all that setup, you can now watch how well different search algorithms guide the Turtlebot3! Test your installation using these steps with the random-walk example. This example implements action execution after each step of action selection. For your search algorithm however, you can use the search-template function and provide only the final, computed plan for execution. However, the overall sequence of launches will remain the same when you run your search algorithms.

1. Start *roscore*.
2. the file ***/search/scripts/server.py*** using command *rosrun search server.py*. You can use the arguments to provide size of the grid, number of obstacles and random seed. For details use ***rosrun search server.py --help***
3. Run the launch file ***maze.launch*** as ***roslaunch search maze.launch***.
4. Run the file ***/search/scripts/move_tbot3.py*** as ***rosrun search move_tbot3.py***.
5. To test the random walk, run ***rosrun search random_walk.py***. To run your search algorithm from the template file, run the template file as ***rosrun search search_algorithm.py*** (you will need to fill in a few "blanks" before doing this). You will need to pass argument **-a** with the name of the algorithm (bfs, ucs, gbfs, astar) you want to run.
6. Now watch your Turtlebot3 move through the maze avoiding obstacles.

Remember to kill the launched servers and nodes and restart from step 2 onwards when you take a new run so that the Turtlebot3 is reset to its initial state prior to running search. This is particularly important for plotting the graph in Q8.