**Search Problems**

**A\* Search**

**Lab**

Below you can find the instructions to complete this Lab question.

Unzip the source code and complete the code implementation for the following:

1. Manhattan Distance function:

def manhattan\_distance(state, goal\_state):

# Function body will go here

* Calculate the absolute differences between the x-coordinates and the y-coordinates of the state and goal\_state parameters.
* Sum the absolute differences calculated in the previous step, which represent the Manhattan distance and return it.

1. Node class:

* Start by defining the class named `Node` with an `\_\_init\_\_` method with these parameters (state, parent, action, cost, heuristic)
* Add the class attributes.
* State represents the state of the node.
* Parent represents the parent node of the current node.
* Action stores the action that led to the current state.
* Cost represents the cost associated with reaching the current state.
* Heuristic stores the heuristic value associated with the current state.

1. Priority Queue Frontier class:

* Start by defining the class named `PriorityQueueFrontier` with an `\_\_init\_\_` method with an attribute `self.frontier` to represent the priority queue.
* Add the `add` method that has one parameter (node) to insert a node into the priority queue. It appends the node to the queue and then sorts it based on the sum of cost and heuristic.
* Add the `contains\_state` method to check if a state is already in the frontier.
* Add the `empty` method to check if the frontier is empty.
* Finally, add the `remove` method to remove and return the node with the highest priority (lowest cost + heuristic) from the frontier.

1. Solve method:

* Start by initializing variables. Create the start node, instantiate a `PriorityQueueFrontier`, and initialize the `explored` set.
* Set up a while loop to iterate until a solution is found or the frontier is empty.
* Check if the current node's state is the goal state. If true, reconstruct the path, store it in `self.solution`, and return.
* Update the explored set with the state of the current node.
* Expand the node by exploring the neighbors of the current node. If a neighbor is not in the frontier or the explored set, calculate its cost and heuristic, create a child node, and add it to the frontier.