CMPT310

Assignment 1

Alex Li

301239152

liyujiel@sfu.ca

# Description

The assignment question is a specific type of Find Shortest Path. It contains start location, goal location in a Matrix(Graph), the most common solution is Dijkstra algorithm. But base on all search algorithms we learned in class, I decide to choose A* algorithm which should have a better performance than common solution. Because A* Algorithm contains both benefit from Dijkstra and Best-first Search Algorithm. The most important part in A* is I need define f(n) by myself, f(n) = g(n) + h(n), as we know g(n) is the value of the location, but it's hard to decide the best heuristic. After searching online, the physical distance between two locations is the heuristic value.

There is one thing I was trying to implement in this assignment is get the shortest path without make pre-process of input which is make it as a graph object. At this point, each time my script searches the current location will try to get 4 different side locations as neighbor and check it still in the range by check location coordinates.

After I got the goal location, another problem I faced is the assignment required return a list of target. It will waste space and hard to keep if I use a list contains visited location through searching process. So, I create a map named came_from, each time I get a better result, I will store the neighbor location as key, and current location as value. Once the search is done, I can track back the target location recursively or return "Not Found".

CMPT310

Assignment 1

Alex Li

301239152

liyujiel@sfu.ca

# Overview of the script

There are 3 methods in my python script:

1. path_find(size, start_loc, goal_loc, values)

   Main function represents A* algorithm, I leave the step by step explanation as comments in my script.

2. heuristic(a,b)

   Heuristic function for calculate heuristic number, I use Pythagorean theorem for calculate distance between nodes.

3. reconstruct_path(came_from,current_node)

   Return a list of locations which use recursively method get current_node from dictionary named came_from.