Task 14 – Lab: Graphs, Paths & Search

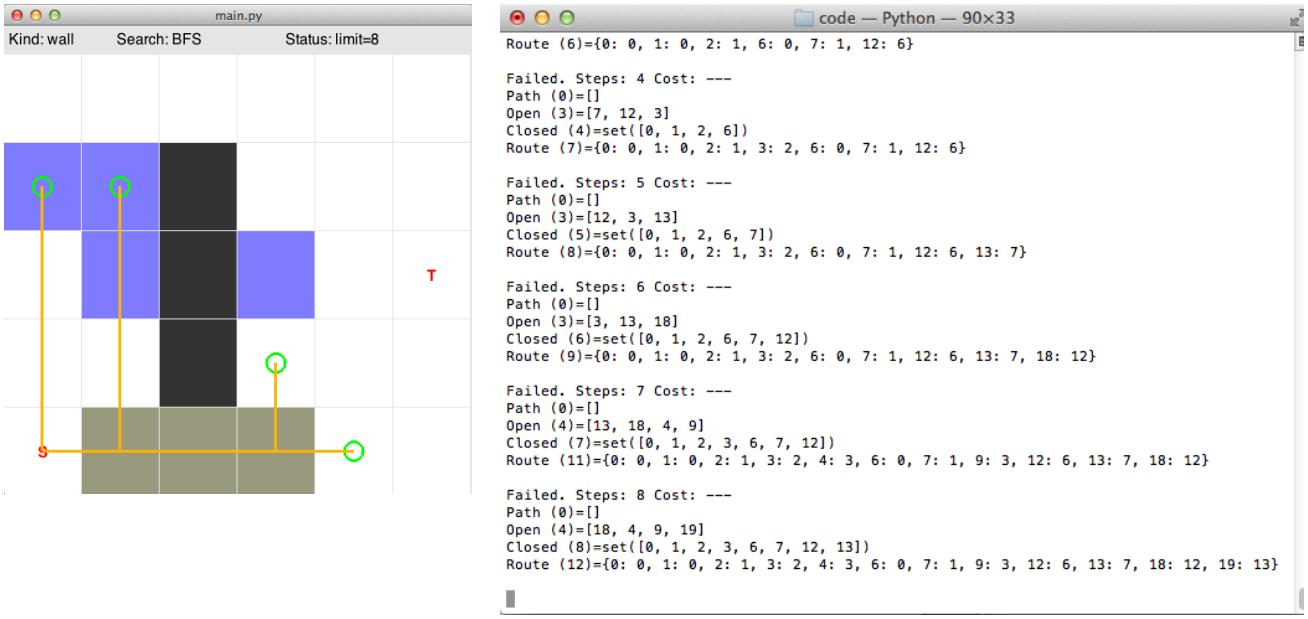
Summary

The goal of this lab is to help you understand and utilise Depth First Search (DFS), Breadth First Search (BFS), Dijkstra's and A* Search algorithms applied to a simple "box" based world. It also helps with Task 15.

Download & Run:

Download the code for this lab from the unit website. Extract to your appropriate work location. The file to run is `main.py`. You will need to specify a map file to load as an argument. There are two simple maps to start with named `map1.txt` and `map2.txt`. For example, to load `map1.txt` you would execute:

```
C:>python main.py map1.txt
```



(a) (b)

Figure 1. In (a) the GUI view of the program for a BFS search limited to 8 steps is shown. In (b) details of the last five step-limited results are shown, with the current open queue yet-to-be looked at, then closed set of nodes already considered, and the route list of the current search results.

Edit the World – Set Boxes, Start and Target

The worlds' boxes can be changed with a combination of selecting the box "kind" (shown in the top left text) and then left-clicking on to a box. There are currently four different box kinds:

1: "clear" (white)
2: "mud" (grey-brownish)
3: "water" (blue)
4: "wall" (black)

Walls are not included (have no edges) in the navigation graph.

For each search a start ("S") and target ("T") box is needed. To change the location of either, first set the kind value by pressing a number key, and then click on a box to set it.

5: "start"
6: "target"

You can allocate start or target to "wall" box (that has no edges), however this will stop any search from being successful!

Search!

Pressing the SPACE key will perform a search using the current map and search mode. However, most changes to the world force a new search to be done immediately for you.

There are currently four different search modes, which can be cycled through using the N and M keys (backwards and forwards respectively). The search modes are:

- "BFS" for Best First search algorithm
- "DFS" for Depth First search algorithm
- "Dijkstra" for Dijkstra's lowest-cost-so-far search algorithm
- "A*" (written as "AStar") for the lowest cost-so-far + lowest-estimated-cost algorithm

At the end of this document is a list of the keys and features for quick reference. You can also read the code directly to understand what you can do, and alter it as you wish.

Search "depth" (the number of search steps taken) can be limited and changed using the UP and DOWN arrow keys. The limit can be removed using the "0" key.

Search Results

When a search is performed you can see several things both visually on the map and in the console text output. For example:

```
Success! Done! Steps: 14 Cost: 18.071
Path (4)=[18, 19, 26, 27]
Open (6)=[3, 9, 15, 16, 22, 28]
Closed (14)=set([0, 1, 2, 6, 7, 12, 13, 18, 19, 21, 24, 25, 26, 27])
Route: …
```

If a successful path has been found, it is shown. Note that a search can be successful (found the target) but not finished (still has alternatives to try). The number of steps and the cost are shown to enable the comparison of algorithms. The route is also printed (but not shown above) which is presented as pairs of (to:from) index values, which is useful to represent the entire search tree and reconstruct the final search path.

Using the UP and DOWN keys, limit the search depth for each search type and observe the changes in the open and closed lists. The numbers match the box numbers, and can be displayed by pressing the "L" key.

Change the location of the Start and Target markers and observe the effect and performance of different searches.

Refer to the lecture notes on graph search and follow the search steps for a small (easy) situation. Look at the code in searches.py and see if you understand the differences for each search method. (They are quite similar.)