Spike: 12
Title: Graphs and Search

Author: Steven Efthimiadis, 1627406

Goals / deliverables:

Demonstrate the use of Dijkstra's (search for item) and A* (search for position):

- Modify the lab 10 code
- Add an agent that follows a path for a successful search result
  - Movement doesn't have to be fancy (same movement as planet wars is good)
- Able to search for an item or points on the map
- Display the cost of the path
- Demonstrate the need for both algorithms

Technologies, Tools, and Resources used:

- Knowledge of python
  - https://docs.python.org/3/tutorial/
- Python Interpreter
  - Visual Studio
    - https://www.visualstudio.com/downloads/
- Knowledge of search algorithms
  - https://artificialintelligentsystems.wordpress.com/category/ai-searching-techniques/page/2/
  - https://www.tutorialspoint.com/artificial_intelligence/artificial_intelligence_popular_search_algorithms.htm

Tasks undertaken:

- Adjust the search file to only have A* and Dijkstra's algorithms (commenting out the code is good idea)
- Add an agent that follows a completed path
  - Store the boxes from the world in the agent
  - When a successful path is made add the path to the agent
  - Once working then add circle (or any other object) that is drawn as the agent moves
- Add the ability to place items
  - If you place an item, make the search algorithm Dijkstra's
  - If you place the target position, make the search algorithm A*
- Changed the status text to display Cost when you have a successful path
- Make a custom map which is cost heavy
  - 20x20 map
  - Lots of mud
  - A couple pools of water
  - A clear path around a pool of water
  - Lots of walls blocking the start and target

What we found out:

- If we look at figure 1.1. We see that Dijkstra has searched nearly the whole map.
  - When it saw that a path was costing too much it would return to a previous node and start searching again from that node
  - It travels around the pool of water and mud because the clear path cost less than traveling through the mud
- If we look at figure 1.2. We see that A* only searched minimal amount of nodes
  - It made a judgement that it didn't care too much that it wasn't costing it more than the estimated heuristic. This means that it would constantly travel through mud.
  - If we change the value of edge cost to 5
    - In figure 1.3 we see that the path looks the similar to the Dijkstra path. Because the estimated heuristic cost was 5. It was searching as many nodes as it could to find the minimal estimated cost. Once it got to the clear path it made a judgement that it

would travel back through the mud because it was less than the estimated heuristic so it travelled back into it

- As we see from the figures. The need for different algorithms is accentual to task required
  - If we need to find an item, we need to have the smallest cost possible because you want to get there quick and safe. Therefore, Dijkstra's algorithm is a good choice. It will search as many nodes as possible to find the most cost effective route while you keep moving.
  - If we need to get to a position, we need to get there quickly and add a bit of dare because there might be obstacles you can handle. Furthermore, A* is a good algorithm to use. Why? Because it will travel through obstacles that are lower than it's predicted heuristic value. So you might be traveling to the same place as the item in figure 1.1 but, due to not being scared of a bit of mud, you travel through it, costing more than Dijkstra.

Open issues/ Risks:

- When you get the starting position, I suggest you don't remove your source location or your path will slowly delete itself.
  - https://ilearn.swin.edu.au/webapps/discussionboard/do/message?action=list_messages&course_id=_178596_1&nav=discussion_board_entry&conf_id=_319510_1&forum_id=_807586_1&message_id=_10187769_1

Notes:

World Keys:
1 – Clear Tile
2 – Mud Tile
3 – Water Tile
4 – Wall Tile
5 – Start Tile
6 – Target Tile
7 – Item Tile

Left Click – Add tiles to boxes

Appendix

Figure 1.1 Dijkstra search for item

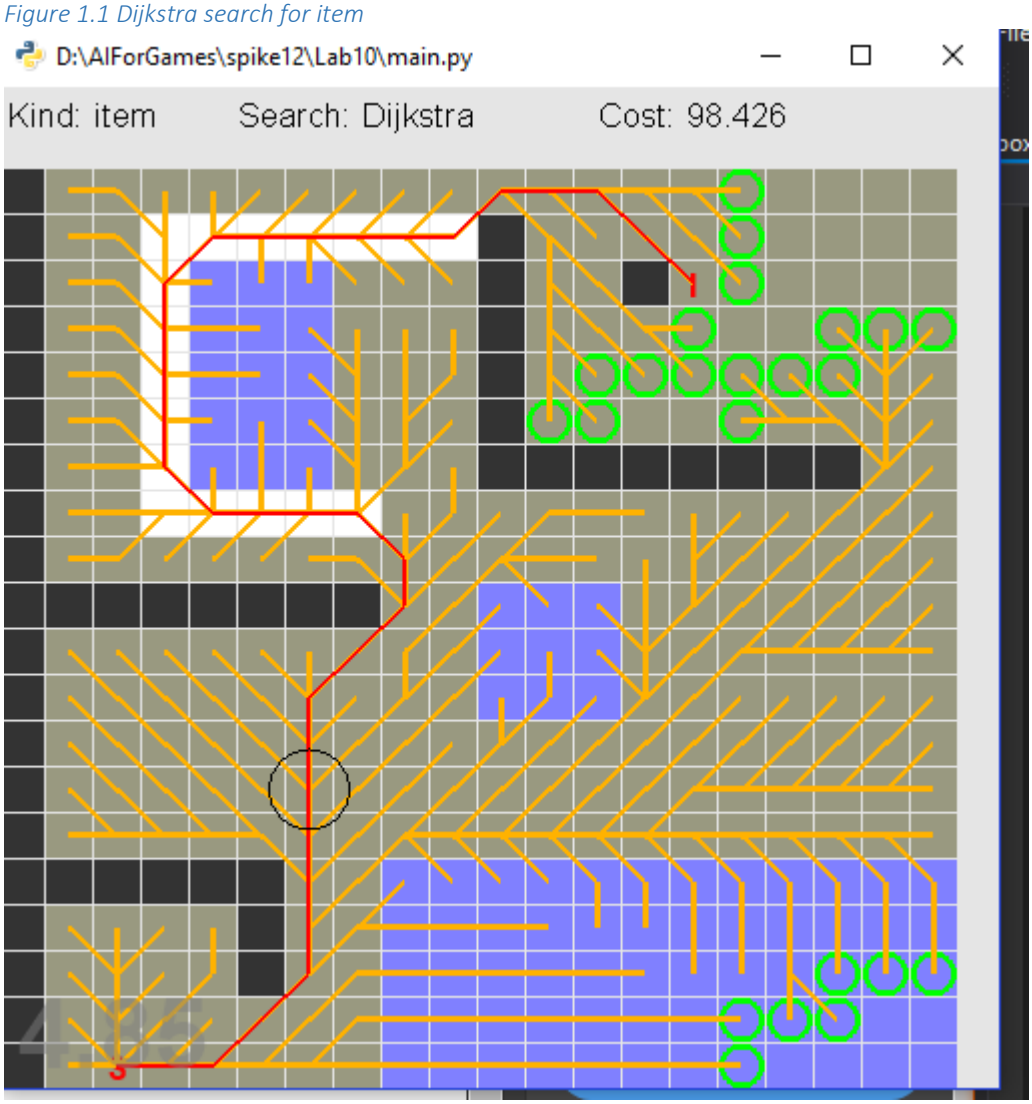

Figure 1.2 A search for position with a minimum edge cost of 10*

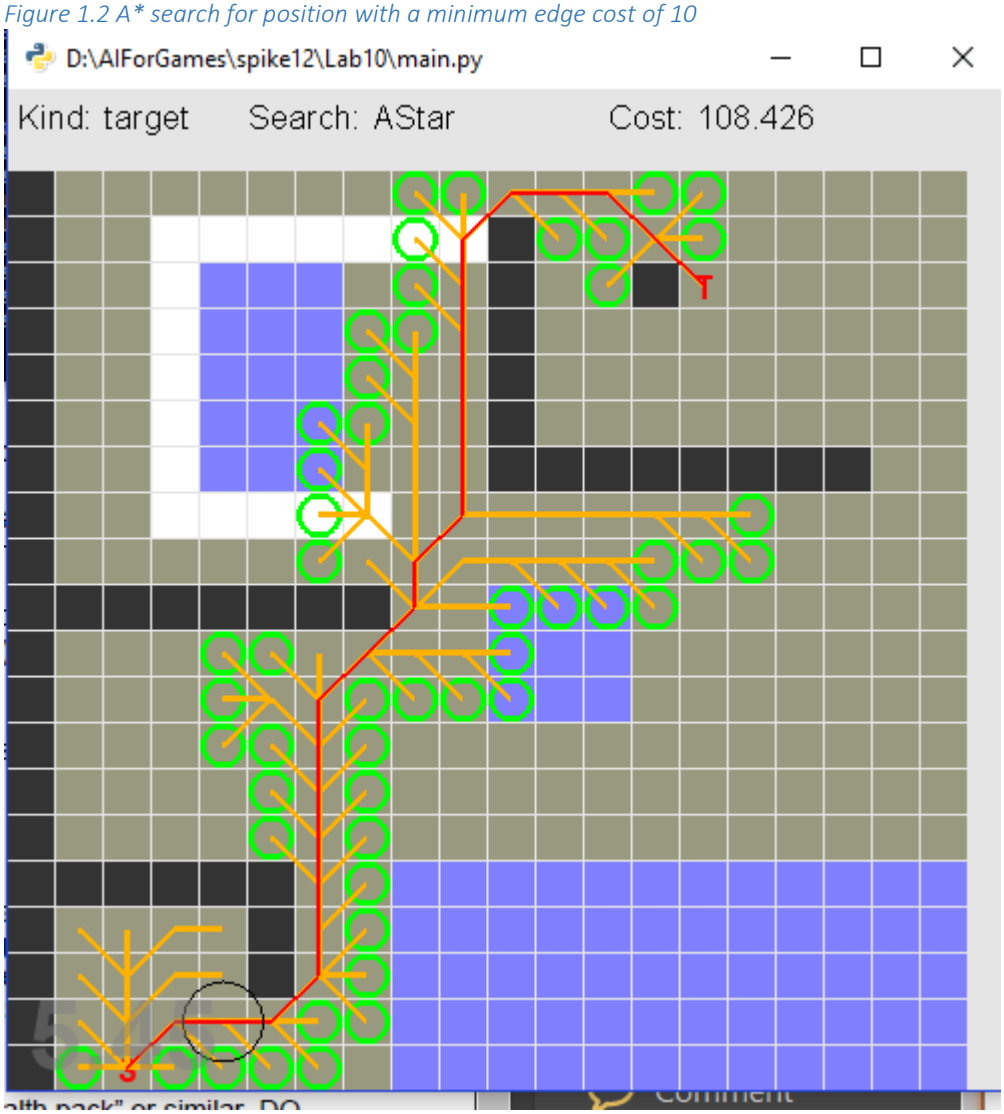

Figure 1.3 A search for position with a minimum edge cost of 5*