Search for routing

A first look at 2 different search algorithms

Robin Andersson 2015-12-22

Introduction

The assignment consisted of 2 tasks:

Implement a search algorithm using the iterative depth-first search.

Implement a search algorithm using the A* search

Both assignments were to be coded in Python.

Initial problem

First a quick look at the 2 different algorithms:

Iterative depth-first search:

This algorithm is an extension of depth-first search, which given by its name searches the depth of a tree before extending its breadth. The iterative depth-first search does this with a mixture of breadth-first search. For each iteration the algorithm searches 1 more level in depth. This means that the algorithm won't search down the full branch, but rather stop when a given level is reached, and continue its search in the other branches.

A* search:

This algorithm focuses on looking at a given cost of traveling from one node top the other. By continuously keeping track of the cost so far and the future cost the algorithm will only extend its search within a given node if the future cost of that node is the cheapest.

The 2 tasks basically have the same foundation, which in turn meant that a properly chosen datastructure would be key in completing them efficiently.

The thought process of choosing the right form of datastructure ended up with 2 main leading phrases:

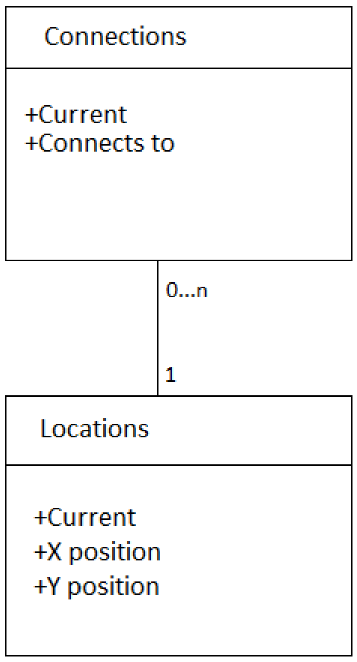
Easy to implement and thus easy to understand

Try to make it as uncomplicated as possible

Another goal also was to learn new datatypes and how they are used, although this wasn't going to be preferred over using datatypes which seemed best for the job.

The first step was to analyze the data that was going to be used for this assignment.

In the document ”connections” all the areas were typed together with another, single area. Example: Bjorkhaga – Alvtomta, Alvtomta – Vasthaga.
It would be suitable to use a datastructure where an area, for example Vasthaga, would be tied together with all of its connections, Solhaga, Alvtomta and Rosta. This would later simplify the way the searchfunction could pick out the correct connected locations.



The document ”Locations” is divided into 2 types of data: the location name and it’s x and y coordinates. Here the most suitable datastructure would have to use a keyword (the location name) and a way of saving the x,y coordinates together.

The diagram on the right hand side shows how “connections” and “locations” are supposed to work with each other.

Process

The datastructures that got chosen in the end were dictionaries, since they are built up of key’s and values, where the values can be used to store a single variable, or a list or a tuple… This flexibility from a single datastructure would make it easier to create reusable functions.

The “connections”-part used the dictionary as a Key = string, Value = [list of strings], as this datastructure needs to keep track of all the places that are connected to a specific area. Therefore the list option is great for this, as when a new connection is added, it’s automatically added to the correct key, or creates a new if no key exists with the given specified area.

The “locations”-part is built up in the same way, except of the use of a tuple as a value instead of a list. This is because there will always be a need of using both x and y coordinates at the same time. There will never be a need of just using 1 or the other.

After the dictionaries were implemented it was quite simple to implement the algorithms using the pseudocode from the lecture slides.

Troubles along the way

The only major problem during the implementation of the algorithms were the part of the iterative depth-first search mechanism which would make sure to search level-by-level through iteration, instead of a typical depth-first search. The problem that occurred was that the algorithm wouldn't wouldn't stop at the intended level, or stop at a certain node and just keep searching from that point on.

This was however quite easily solved when the writer of this report found the extra slides of how this algorithm should be made.