Route Search in Logistics problem

Introduction and some programming hints

Departamento de Informática
Universidad Carlos III de Madrid

Final assignment - 2018-19

Summary

- We'll program a practical application of search
- Algorithms are provided by the package SIMPLE-AI
- Interface software (game) is provided by additional code
- All the student work is done by modifying files in the student folder
- Experimentation is very important: different algorithms, different maps, impact of the heuristic, etc. Coding is only part of the task!
- You must deliver through Aula Global before May 12th: a single compressed file for each group (max. 2p) that contains both essay and code (see the project definition for details):
  1. Essay, PDF document with description of the solution, experiments and conclusions
  2. Software in a separate folder

Software

- **Plataform**:
  - Reference platform is Linux
  - Python version is 2.7
  - `simple-ai`: AI algorithms
  - `game`: Interface libraries
  - `student`: Student files
- Dependencies:
  - `simple-ai`: Requires *Flask* for the web interface (installed in the lab)
  - `game`: Requires the package *pygame* (installed in the lab)
  - `pydot`: Installed in the lab, or install from command-line: `pip install pydot`

How does it work

1. When `startGame.py` is executed, the map is loaded, debug information is printed to the console, and the interface is initialized using the information in `config.py`. A `GameProblem` object is created and the local variables (such as AGENT_START, POSITIONS, etc.) are initialized. Remember that in order to use these variables, they must be prefixed by `self`.
2. `setup` is called: you must create the initial state and define the search algorithm that will be used to find a path. You may also define your final state here, but you must check it explicitly in the method `isGoal`.
3. `actions` is called with your initial state as a parameter. You must generate a **list** of actions. If actions are in the list of movement actions (North, East, etc.), they will generate moves in the map when the path is followed, but you can use additional actions.
4. `result` is called when the search algorithm expands the current node; you must return the new state that is reached when executing the specified action in the current state. The initial code returns “0” (which is probably not a meaningful state).

The search algorithm will select a node among the expanded ones (fringe) and repeat from step 3.

How does it work (II)

- `isGoal` is called when the algorithm needs to check whether a state is the goal. You **must** change this method so it does not always return a **True** value, otherwise the above methods are not called.
- `cost` and `heuristic` have to be implemented or not depending on the part of the project (ref the doc.).
- `printState` and `getPendingRequests` are called when tracing the path on the map, in order to display the information on the message area and change the icon that represents the number of pending deliveries.

Other issues

Recommendation: start with a simplified version of the problem that consists just in **moving from the start position to a certain end position**. Then add the restrictions (blocked positions), then load/unload, etc.

Important: some implementation restrictions on the algorithms require the **state** to be coded as an immutable object; this means you have to use a **tuple** `( element, element, ...)` and **not** a list or dictionary.

- Initially you will only change the configuration file to switch the map you want to load. You can also modify the start position of the agent.
- After loading the configuration file, all the information you need is loaded in the
- Use the method `getAttribute(<position>,<key>)` to check the values in the attribute section of the map for any position, to decide the type of terrain, number of orders, etc.
- For the advanced problem you may add new terrain types following the template for the ones listed there.

Python vs. Java

- Important: Code blocks must be **must be indented**. Use either TAB or 4 spaces, but **don't mix both**
- Comment lines start with `#`, comment blocks are surrounded by three single quotes (`"'`).
- Many errors will appear in execution time, when functions or methods are called the first time.
- Logic operators are: `and`,`not`,`or`,`==`
- Strings are defined using single or double quotes. String concatenation is allowed using '+'. Access substrings, tuples and lists allow slice syntax (`variable[start:end+1]`, first element is 0).
- Functions are invoked with operators and may return a list of results: `res1, ..., resn = function(parameters)`
- Methods are invoked on objects using the dot syntax: `object.method(parameters)`
- Use the function `print()` with a single argument `print("String")`. Format strings with the format method `<string>.format(<list>)`.

```
print ("Item numer \{0\} es \{1\}".format(1,"A result"))
```

There is plenty of information online: `https://wiki.python.org/moin/`

Python frequent errors

- **Problem**: ImportError: No module named py

  Do not use the extension ".py" with `import`
- **Problem**: NameError: name 'MYVARIABLE' is not defined

  To use a variable you must use the name or alias of the module, or `self` inside an object: `module.MYVARIABLE` , `self.MYFUNCTION`,etc.
- **Problem**: TypeError: 'dict' object is not callable

  To address the value in a `dictionary` we use square brackets `dict["key"]` (not parenthesis "`dict("key")`")
- **Problem**: ValueError: too many values to unpack

  In a multiple assign we must match the elements to be assigned with the number of variables `a,b = (1,2)`
- **Problem**: AttributeError: 'list' object has no attribute 'length'

  The length of a list is returned by a function `len(LIST)`, **not** a method `LIST.len()`
- **Problem**: We've made changes in a file but the result has not changed

  After saving an imported file, we must reload the module: `reload(MODULE)` or `reload(ALIAS-MODULE)`