Artificial Intelligence for Robotics – Lab

Prof. Dr. Erwin A. Prassler: *erwin.prassler@h-brs.de*

Maximilian Schöbel: *maximilian.schoebel@h-brs.de*

Patrick Nagel: *patrick.nagel@h-brs.de*

Assignment 3

Due Date: Tuesday, 6.11.2018, 08:00

1. (2 Points) The following pseudo-code describes the generic tree-search and graph-search algorithm. Briefly answer the questions below.

```
function TREE-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        expand the chosen node, adding the resulting nodes to the frontier

function GRAPH-SEARCH(problem) returns a solution, or failure
    initialize the frontier using the initial state of problem
    initialize the explored set to be empty
    loop do
        if the frontier is empty then return failure
        choose a leaf node and remove it from the frontier
        if the node contains a goal state then return the corresponding solution
        add the node to the explored set
        expand the chosen node, adding the resulting nodes to the frontier
            only if not in the frontier or explored set
```

Abb. 1: Pseudocode of the genereal tree-search and graph-search algorithms from Russel & Norvig

- What is the fundamental problem, when applying an uninformed search algorithm on graphs?
- How does the general graph-search algorithm address this problem?
- Is the distinction between tree and graph-search specific to each individual search-algorithm or can all search algorithms be implemented in both flavours?
- Given a problem, that is solvable by a search algorithm. How do we chose between tree and graph search?

2. (6 Points) On LEA you will find three text files, each representing a map for this week's programming assignment. A search agent is supposed to explore each of these three maps using

a) **breadth-first search**

b) **depth-first search**

c) **iterative-deepening depth-first search**

The maps contain the following information:

- Each character in the text file represents a "cell" in the map.
- (`*`) Cell contains a goal.
- (Space) Cell is free.
- (`s`) Initial position of the agent.
- Any other character represents an obstacle.

Your task is to implement the above mentioned search-algorithms to make your agent explore and find each dirt cell. The agent must follow these rules:

- The agent can move through explored regions freely and can explore only one cell at a time.
- The agent can only move to the left, right, up or down from the current position (not diagonal). The surrounding cells should be considered as the children at the current agent's position.
- The agent does not have previous knowledge about the environment (such as dirt positions or obstacles) so it has to "explore".
- The agent cannot move through obstacles and the map is closed.

Make your implementation provide terminal output, so that the functioning of your program is easily readable, e.g. by printing the map with all explored nodes marked for each step of exploration. To circumvent compatibility issues, **provide that output in text-files along with your submission** for each of the implemented search algorithms.

TIP: You can re-route the output of your program to a file by using the `>` operator. The file will be created if not existing or overwritten if already existing. The `>` operator takes the output of a command-line program and puts it into the specified file. For example:

```
python3 my_python_script.py > output-file1.txt
```

To append output to a file (and not overring its content) use the `>>` operator instead.

3. (4 Points) Visualize the search-tree that your algorithm spans along the map. You can use the following unicode-characters to do so.

U+253C ┼
U+2502 │
U+2500 ─
U+2534 ┴
U+252C ┬
U+251C ├
U+2524 ┤
U+250C ┌
U+2510 ┐
U+2514 └
U+2518 ┘
U+2574 -
U+2575 ╵
U+2576 -
U+2577 ╷

To print or assign a unicode-character to a variable use the following syntax in python 3:

```
my_agent_map[x][y] = ‘‘\u253C’’
print(‘‘\u2577’’)
```

4. (1 Point) What are the properties of Iterative-deepening depth-first search? Is it optimal and does it find shortes paths in your map? Explain why / why not.