Behavior Trees

VACUUM CLEANING ROBOT'S ROAMINGS (100 Points)

This new type of robotic vacuum cleaner has quite simple reflex rules. It will always check the battery level first. If the level is below 30%, it will plan a path to its charging base ("home"), go there, and start the docking procedure. If the battery is at a sufficient level, it will start the function it was commanded to perform. There are two available commands:

1. Spot cleaning: it will perform a 20s intensive cleaning in a specific area.
2. General cleaning: go around the room and vacuum dust until the battery falls under 30% or it completed the task. If the dust sensor detects a particularly dirty spot, the robot will perform a 35s spot cleaning.

Your implementation should accept a blackboard object as input (a regular hash map or dictionary). The blackboard contains the following elements:

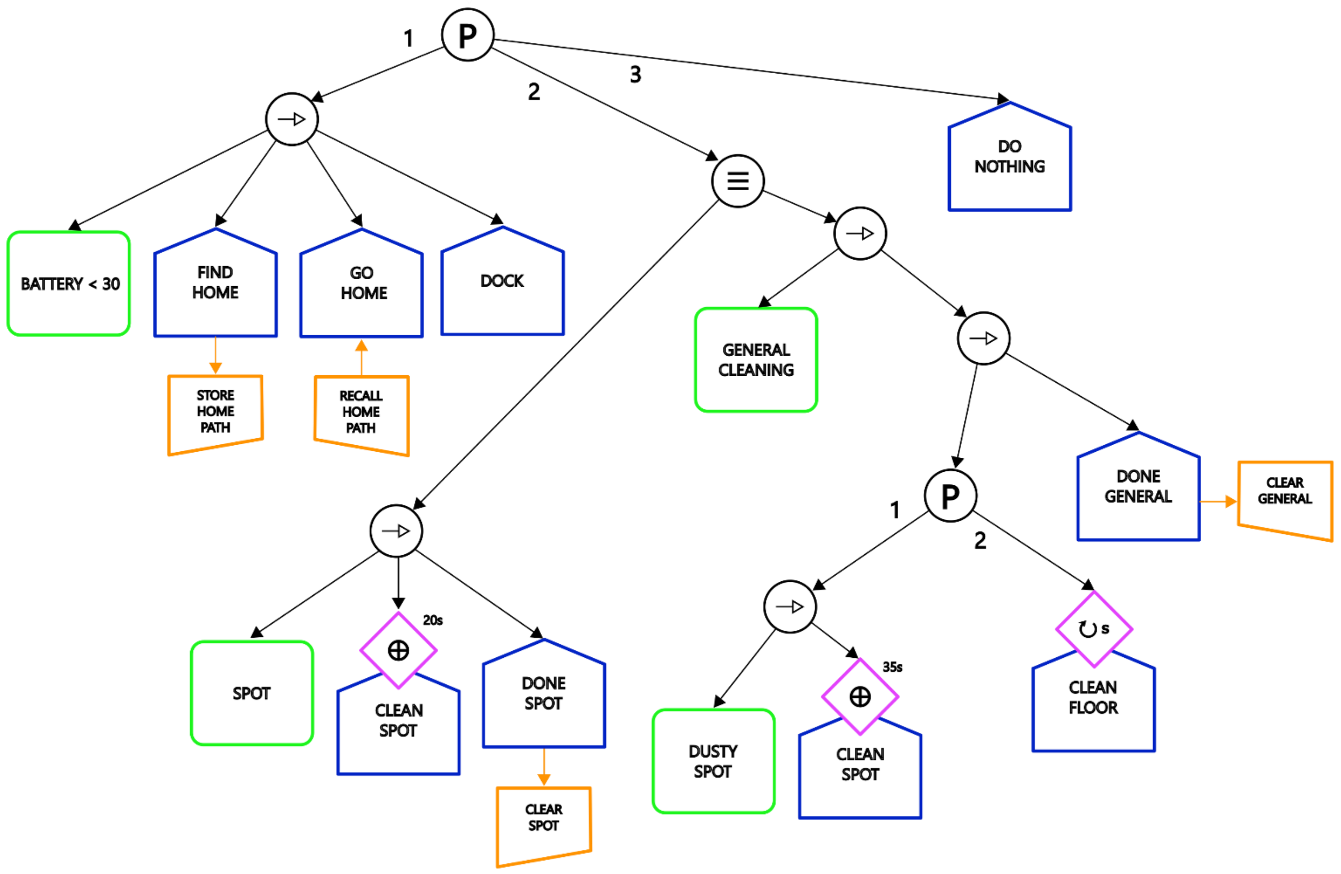
1. **`BATTERY_LEVEL`**: an integer number between 0 and 100.
2. **`SPOT_CLEANING`**: a Boolean value – **`TRUE`** if the command was requested, **`FALSE`** otherwise.
3. **`GENERAL_CLEANING`**: a Boolean value – **`TRUE`** if the command was requested, **`FALSE`** otherwise.
4. **`DUSTY_SPOT`**: a Boolean value – **`TRUE`** if the sensor detected a dusty spot during the cycle, **`FALSE`** otherwise.
5. **`HOME_PATH`**: The path to the docking station.

As specified in the tree, **`SPOT_CLEANING`** and **`GENERAL_CLEANING`** should not change state until the command has been completed. The tree evaluation should be called several times. For the sake of simplicity, we will assume that each call is at 1s intervals. Certain tasks should return **`RUNNING`** if they have not completed the job yet, and last for the specified number of cycles (20 or 35 cycles).

The **`CLEAN_FLOOR`** task will fail when there is nothing more to clean. The result of the task can be determined at random (with a low probability of failure) or asking the user (maybe every few evaluation cycles).

The goal of this assignment is to implement the provided behavior tree. As we talked about in class, trees can be represented with concatenated IF-THEN-ELSE rules. However, the assignment requires a proper object-oriented design, in which a hierarchy of class definitions for the node types (basic node, composites, tasks, conditions, and decorators) is defined. Each specific task, condition, and decorator in the tree in the figure should be implemented as a descendant of the hierarchy.

I strongly suggest implementing this agent as a basic reflex agent. The sensors can be simulated with simple random values or with an input from the user. The state of the environment (the percepts) is stored in the blackboard before being passed to the behavior tree evaluation (condition-action rules block). Except for **DONE GENERAL** and **DONE SPOT**, none of the other tasks will need to be implemented. A simple print statement with the name of the task and the state (SUCCEEDED, FAILED, RUNNING) will be sufficient. **DONE GENERAL** and **DONE SPOT** will set to **FALSE** the corresponding values in the blackboard.



SUBMISSION

Python or C++ are the preferred implementation languages. If you are writing in C++, please include a Makefile as well as any other instructions for compilation. For Python, simply provide a plain PY file (no Jupyter notebook).

Your solution may make use of any numerical libraries for pre-processing, fundamental calculations (i.e., linear algebra) and visualization. However, the core portion of your solution must be implemented from scratch.

Submit your solution via Canvas and include a README file that clearly explains its assumptions.