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 sufficient, 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 completes the task. If the dust sensor detects a particularly dirty spot, the robot will perform a 35s spot cleaning.

The blackboard must contain at least the following elements, but you can add more if your implementation requires it:

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.

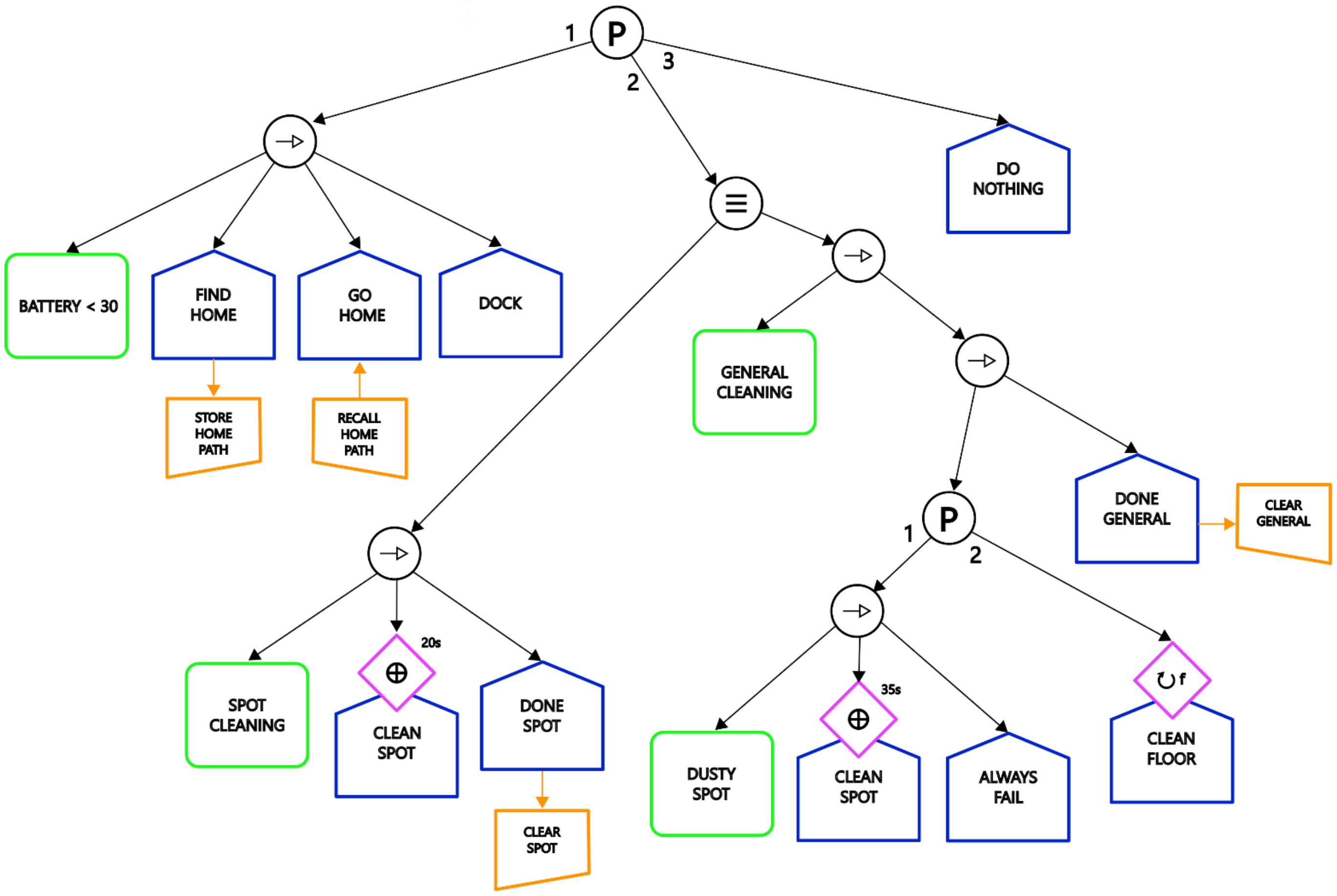
As specified in the tree, **`SPOT_CLEANING`** and **`GENERAL_CLEANING`** should not change state until the command has been completed. The tree evaluation should be called several times. For the sake of simplicity, we will assume that each call is at 1s intervals. Certain tasks should return **`RUNNING`**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

The value of **`DUSTY_SPOT`** sensor should never be changed in the behavior tree itself, but it should be changed in the main body of your solution before every evaluation (see the main body of the program for more details). Regardless, you are free to simulate the sensor value generator in any way you want (i.e. via a question to the user or a random generator).

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 by asking the user (maybe every few evaluation cycles).

The goal of this assignment is to implement the provided behavior tree. Each specific task, condition, and decorator in the figure should be implemented as a descendant of the hierarchy provided by our code.

I strongly suggest implementing this agent as a basic reflex agent. The sensors can be simulated with simple random values or input from the user. The state of the environment (the percepts) is stored on 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 the corresponding values on the blackboard to FALSE.



SUBMISSION

Python or C++ is the preferred implementation language. For both languages, we are providing a partial implementation. Study the code and implement any missing task or condition. Do not change the core class definitions and the basic mechanisms (anything in the `bt_library` directory). Feel free to change anything else. For Python, provide a plain PY file (no Jupyter Notebook).

Submit your solution via Canvas and include a README file that clearly explains its assumptions. Make sure that you also submit the partial implementation of choice. Your code should run immediately without any additional steps on our part.