TAU - Advanced Topics in Programming, Semester B 2024

Assignment 3

Note:
This document is in DRAFT mode till Aug-4th. While in draft mode, it may have significant changes without notice and without highlighting them.
After the draft period there could still be changes but then they would be highlighted in the document.
During the draft period, you are mostly welcomed to add questions directly into this document, as comments. Please assign your comments and questions to kirshamir@gmail.com so I will get an email when you add them. Then please follow my answer and if it resolves your comment or question please close it.
After the draft period, commenting on the document will be closed and all questions on the assignment would be submitted in the relevant forum in the course site!

Submission deadline: Aug-25th, 23:50.
Submission instructions - TBD

Assignment 3 - Requirements and Guidelines

In this assignment, you are requested to generate three separate projects: two projects for two separate algorithms and one project for your simulator.
The algorithm projects will generate .so files (shared objects, library files).
The simulator project will generate an executable.

There are no changes to the House file and its rules.

The output file does have additions, colored in blue below. An added "InDock" should have a value of either the text TRUE or FALSE, for whether the robot is in the docking station when the simulation ended (TRUE) or not (FALSE).

Output File - TEXT file format
Note: The file name shall be: <HouseName>-<AlgorithmName>.txt
NumSteps = <NUMBER>
DirtLeft = <NUMBER>
Status = <FINISHED/WORKING/DEAD>
InDock = <TRUE/FALSE>
Score = <NUMBER>
Steps:
<list of characters in one line, no spaces, from: NESWsF – the small s is for STAY, NESW are for North, East, South, West. F is for Finished - only if the algorithm actually returned "Finished">

Note: number of characters in the last line of the file should be equal to the NumSteps value, excluding a final F if appears (F is not counted as a step).

Score
Algorithms would now have a score, based on the following formula:
If DEAD => Score = MaxSteps + DirtLeft * 300 + 2000
Otherwise, If reported FINISHED and robot is NOT InDock
    => Score = MaxSteps + DirtLeft * 300 + 3000
Otherwise => Score = NumSteps + DirtLeft * 300 + (InDock? 0 : 1000)
Lower score is better.

API

In addition to the API provided in assignment 2, you will have now an addition for self registration of the algorithms:

```
//-------------------------------
// common/AlgorithmRegistrar.h
//-------------------------------
using AlgorithmFactory = std::function<std::unique_ptr<AbstractAlgorithm>()>;

class AlgorithmRegistrar {
    class AlgorithmFactoryPair {
        std::string name_;
        AlgorithmFactory algorithmFactory_;
    public:
        AlgorithmFactoryPair(const std::string& name, AlgorithmFactory algorithmFactory)
            : name_(name), algorithmFactory_(std::move(algorithmFactory)) {}
        // NOTE: API is guaranteed, actual implementation may change
        const std::string& name() const { return name_; }
        std::unique_ptr<AbstractAlgorithm> create() const { return algorithmFactory_(); }
    };
    std::vector<AlgorithmFactoryPair> algorithms;
    static AlgorithmRegistrar registrar;
public:
    // NOTE: API is guaranteed, actual implementation may change
    static AlgorithmRegistrar& getAlgorithmRegistrar();
    void registerAlgorithm(const std::string& name, AlgorithmFactory algorithmFactory) {
        algorithms.emplace_back(name, std::move(algorithmFactory));
    }
    auto begin() const { return algorithms.begin(); }
    auto end() const { return algorithms.end(); }
    std::size_t count() const { return algorithms.size(); }
    void clear() { algorithms.clear(); }
};

//-----------------------------------
// simulator/AlgorithmRegistrar.cpp
//-----------------------------------
AlgorithmRegistrar AlgorithmRegistrar::registrar;

AlgorithmRegistrar& AlgorithmRegistrar::getAlgorithmRegistrar() { return registrar; }

//------------------------------------
// algorithm/AlgorithmRegistration.h
```

```
//-----------------------------------
struct AlgorithmRegistration {
    AlgorithmRegistration(const std::string& name, AlgorithmFactory algorithmFactory) {
        AlgorithmRegistrar::getAlgorithmRegistrar()
              .registerAlgorithm(name, std::move(algorithmFactory));
    }
};

#define REGISTER_ALGORITHM(ALGO) AlgorithmRegistration \
   _##ALGO(#ALGO, []{return std::make_unique<ALGO>();})
```

Code example: https://coliru.stacked-crooked.com/a/695a7e8a343ca22b
We will explain this code in class.
Your tasks:
Your *main* needs now to run a simulation that includes several houses and several algorithms.
The command-line arguments are now changed to the following:
-house_path=<housepath>
-algo_path=<algopath>
The two command line arguments may appear in any order, you are not required to have support for spaces around the equal sign.
For the house files, the program will try to open all files with suffix .house
For the algorithm files, the program will try to open (dlopen) all files with suffix .so (make sure to also close them with dlclose at the end of the run to avoid leaks).
If a certain file is invalid, proper errors will be written into a matching output file (<housename>.error or <algoname>.error) in the current working directory and the file shall be ignored.
In case any of the two is missing, the program shall look for the files which didn't get folder location in the current working directory.
The program shall run simulations for all the combinations of house⇔algorithm (the "cartesian product" of the two).

Algorithm
In this assignment you should submit two algorithms, each in a separate folder and each with a separate cmake. Both should create an .so file. Both should have a class name based on your IDs, add at the end any comment that you want (such as "A" and "B") to distinguish between the two. The .so filename shall be similar to the algorithm names.
We will conduct a competition between all algorithms, so try to come up with two different algorithms which are both smart.
The algorithm goal is to clean the house with a minimal amount of steps and return Finish when back to the docking station, after all accessible locations are clean. The score formula is given so you can try to optimize based on that knowledge.

Note: your algorithm in this assignment can be non-deterministic (i.e. use some degree of randomness), if you feel that it serves your goal.

No *new* and *delete* in your code
Same as in assignment 2. See explanations there.

Error Handling
Exact same instructions as in assignment 1.
+ Please note the additional requirement for a separate error file per each bad house file / bad algorithm file. A bad algorithm file is an .so that cannot be opened or if the number of algorithms in the registrar is not increased after opening the .so, or if the factory in the .so doesn't create a valid algorithm. A bad house is if the house file does not meet the rules as set in assignment 2.

Running the Program

myrobot -*house_path*=<housepath> -*algo_path*=<algopath>

Order of arguments may change.
If missing a path argument, the program should look for the files in the current working directory.

Additional Requirements

1. The simulation shall generate a summary.csv file with score results, rows are the algorithms, columns are the houses. Add the names for both the houses and algorithms. Each cell shall hold the score for the pair house⇔algorithm.
2. Allow the simulation to run using multithreading, so you can run more than a single pair. Add an additional command-line argument -*num_threads*=<NUMBER> shall be supported, if not provided assume *num_threads*=10.
3. Add a reasonable timeout per each pair house⇔algorithm run, e.g. 1ms*MaxSteps. If this timeout is reached and the simulation didn't finish, the score for this run shall be: MaxSteps * 2 + InitialDirt * 300 + 2000. Even if the thread will return later with another score, the other score will be ignored. Spawn another thread instead of the "stuck" thread. OPTIONAL: if you can terminate the "stuck" thread, do that, otherwise you may want to lower its priority to IDLE. The goal is that if one of the algorithms is very slow (e.g. due to too deep recursions), it will not delay or stuck the simulation.
4. Add an additional command-line argument -*summary_only* with this flag, the simulation will generate only the summary.csv file and error files, but will not generate the other output files per house⇔algorithm pairs.

Bonuses
You may be entitled for a bonus if winning one of the first positions in the class algorithm competition (no need to add a bonus.txt file for that).
Other interesting additions may be also entitled for a bonus (add a bonus.txt file).

IMPORTANT NOTE:
1. In order to get a bonus for any addition, you MUST add to your submission, inside the main directory, a *bonus.txt* file that will include a listed description of the additions for which you request for a bonus.
2. If you already asked for a bonus in assignment 1 or 2 for a certain addition you shall not ask for the same bonus again. If you added something, focus in your bonus.txt file on the addition.

<u>Q&A</u>

Q: Should the FINISH step be counted as a step? Example: the simulation starts and the robot returns FINISH on the first call to nextStep().
A: The FINISH step should not be counted as a step. In the case where the simulation starts and the robot returns FINISH on the first call to nextStep(), NumSteps in the output file and for the score function shall be 0.

Q: In case of DEAD situation, how should we count NumSteps? Example: MaxBattery is 1, simulation starts robot moves out of docking and gets DEAD.
A: The step that the robot cannot perform, as it is DEAD, should not be counted as a step. In the case where the simulation starts and the robot is DEAD after one step, NumSteps shall be 1. Note that the score formula in this case is using anyhow the MaxSteps (as a sanction for the algorithm reaching DEAD situation).

Q: In case MaxBattery is 1, what is the expected best behavior for the algorithm.
A: The expected best behavior for the algorithm in such a case is to immediately return FINISHED on the first call to nextStep(), to avoid DEAD case. If the robot is in the docking station, it should just stay there.