≡   🔍 (https://profile.intra.42.fr/searches)   **amatilda**

(https://profile.intra.42.fr)

SCALE FOR PROJECT LEM_IN (/PROJECTS/LEM_IN)

You should evaluate 2 students in this team

★

Git repository

`vogsphere@vogsphere.21`   📋

Introduction

Please respect the following rules:

- Remain polite, courteous, respectful and constructive
throughout the correction process. The well-being of the community
depends on it.

- Identify with the person (or the group) graded the eventual
dysfunctions of the work. Take the time to discuss
and debate the problems you have identified.

- You must consider that there might be some difference in how your
peers might have understood the project's instructions and the
scope of its functionalities. Always keep an open mind and grade
him/her as honestly as possible. The pedagogy is valid only if
the peer-evaluation is conducted seriously.

Guidelines

- Only grade the work that is in the student or group's
GiT repository.

- Double-check that the GiT repository belongs to the student
or the group. Ensure that the work is for the relevant project
and also check that "git clone" is used in an empty folder.

- Check carefully that no malicious aliases was used to fool you
and make you evaluate something other than the content of the
official repository.

- To avoid any surprises, carefully check that both the correcting
and the corrected students have reviewed the possible scripts used
to facilitate the grading.

- If the correcting student has not completed that particular
project yet, it is mandatory for this student to read the
entire subject prior to starting the defence.

- Use the flags available on this scale to signal an empty repository,
non-functioning program, a norm error, cheating etc. In these cases,
the grading is over and the final grade is 0 (or -42 in case of
cheating). However, with the exception of cheating, you are
encouraged to continue to discuss your work (even if you have not
finished it) in order to identify any issues that may have caused
this failure and avoid repeating the same mistake in the future.

Attachments

⬜ generator (/uploads/document/document/499/generator)

⬜ Subject (https://cdn.intra.42.fr/pdf/pdf/6161/lem-in.en.pdf)

Mandatory part

Reminder : Remember that for the duration of the defence, no segfault, nor other unexpected, premature, uncontrolled or unexpected termination of the program, else the final grade is 0. Use the appropriate flag. This rule is active thoughout the whole defence.

Prerequisites

If something here is missing, the defence is finished and final grade is 0.

- Nothing in the git repository.
- Check that the author file is at the root of the repository and formatted as explained in the subject.
- The norm is not respected, the norminette is proof.
- Cheat (The student can not explains her/his code) => Flag Cheat
- A prohibited function is used.
- The program is called lem-in
- The program must be able to read an ant-farm on the standard output.
(For example: "./lem-in < ant_farm.txt")

☑ Yes   ✕ No

Error management

The program must manage errors properly on the standard output. Remain open minded when considering the design implemented for that section, and most importantly with regards to the error messages. Don't be picky just make sure it's consistant. Basically, we mean that, if the project displays either something different than "ERROR" or if the choice was made to display nothing as an answer when the ant-farm has no ants, you can accept it.

Missing information

- Evaluate the implementation of error management when there is no room.
- Evaluate the implementation of error management when there is no ant.
- Evaluate the implementation of error management when there is no ##start / ##end.

⊘ Yes   ✕ No

Comments and commands

- The program accepts ant-farms with comments.
- The program accepts ant-farms with commands other than ##start ##end.

⊘ Yes   ✕ No

No possible solution

Evaluate the implementation of error management when there is no possible
solution with the given ant-farm.

⊘ Yes   ✕ No

Output format

The output format must be observed.

Ant-farm composition.

The ant-farm's composition is displayed on the standard output,
as well as commands and comments.

⊘ Yes   ✕ No

Ants movements

The movements are printed with the right format:
- 1 line per turn
- N movements per turn
- A movement is defined as follow "Lx-y" where x is the ant's
number and y a room's name

⊘ Yes   ✕ No

Explainations

Algorithm explaination

Here the student will have to explain her/his approach and justify her/his choices
in research and implementation of her/his algorithm.
- Why is it adapted to the problem?
- How does it work?
Has the corrector understood how the algorithm works?
This step is necessary for the validation of the project.
- Take all the time you need to understand.
- Ask your questions that are useful for understanding.

    ✓ Yes   ✕ No

Test the algorithm

Path validity

Throughout this section the program must go off the valid paths.
It is up to you to check the validity of the outputs.
If only one of these outputs is not valid, check No and the defense stops here.

Valid means:
- At each line, only one movement per ants
- That before a move the room targeted by the ants is free
(there is no ant inside)
- That ants move in rooms within range
(connected by a tube to the current room)
- That at the end of the program all ants are in ##end

If none of your tests obtain an invalid solution, check Yes.

Does the program always provide a right solution?

    ✓ Yes   ✕ No

Flow management

Does the program use several paths when they do not overlay each other and
that this is necessary?
You must use the generator given as an example with the --flow-one option, then
--flow-ten, then --flow-thousand.
The generator, indicates the number of lines expected, any difference with
this expectation shows a poorly optimized algorithm.
These cases are easy to implement and do not require any elaborate algorithms.
Therefore, we will only tolerate a difference of a maximum of 3 lines here.
Otherwise it means that the paths are misused, or that not all of them are found by the algorithm.
The student has then misunderstood the subject, check No and the correction stops here.

    ✓ Yes   ✕ No

Time complexity

Test the program with maps generated by the generator with the --big option,
then test it with the --big-superposition option.
Measure the execution time of the program, and note on several tests.
- 5 --> Well done, perfect, the program always runs around 3 seconds or less.
- 4 --> very well, the program runs in 6 seconds maximum.
- 3 --> the program runs in 9 seconds maximum.
- 2 --> lack of optimization, the program runs in 12 seconds maximum.
- 1 --> very slow, however, the program runs in 15 seconds maximum.
- 0 --> ...the program sometimes runs in more than 15 seconds, the subject is not respected, the defense stops here.

Rate it from 0 (failed) through 5 (excellent)

Algorithm's accuracy

Measure the accuracy of the program's results:
The maps generated by the generator with the --big-superposition option are used here
to check the proper management of the path overlay problem.
The generator indicates the number of lines expected as comments.
This is a complex and important problem in graph theory,
processing it correctly is required.

Attention! The maps generated by the generator are by no means exhaustive
for these tests, you are invited to test the algorithm with maps of your creation.
Use your imagination to test the consistency and the accuracy of the algorithm.

- 5 --> everything is absolutely perfect, well done, the algorithm always gives a number of lines equal or lower than that of the generator.
- 4 --> optimized algorithm but the result is sometimes one to two lines higher than the generator.
- 2 --> good algorithm but the result is sometimes 3 to 10 lines higher than the generator.
- 2 --> On some maps, the number of difference lines increases dramatically.
- 1 --> the algorithm is designed to manage the superposition of paths but the results are very far from the objectives.
- 0 --> the algorithm does not handle path overlay, the subject is not respected, the defense stops here.

Rate it from 0 (failed) through 5 (excellent)

Bonus

Bonuses

In this section you can evaluate up to 5 operational and well implemented bonuses.

Bonus example:
- Graphic visualizer -> if you find it very well made, you can give more than 1 point for it.
- ...

Rate it from 0 (failed) through 5 (excellent)

Ratings

Don't forget to check the flag corresponding to the defense

✔ Ok   ★ Outstanding project

📄 Empty work   📑 Incomplete work   💬 No author file   💀 Invalid compilation   📕 Norme

📑 Cheat   ♟ Crash   👥 Incomplete group   💧 Leaks   🚫 Forbidden function

Conclusion

Leave a comment on this evaluation

Finish evaluation