Introduction

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

- Identify with the person (or the group) evaluated 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 and only if 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 evaluating and the evaluated students have reviewed the possible scripts used to facilitate the grading.

- If the evaluating 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.

- Remember that for the duration of the defence, no unexpected, premature, uncontrolled or unexpected termination of the program is allowed, else the final grade is 0. Use the appropriate flag.
You should never have to edit any file except the configuration file if it exists.
If you want to edit a file, take the time to explicit the reasons with the evaluated student and make sure both of you are okay with this.

Attachments

subject.pdf mlx_CLXV-2.2.tgz output_validator.py

mlx-2.2-py3-ubuntu-any.whl mlx-2.2-py3-fedora-any.whl

Submitted files and Norm

Check if all expected files are present in the Git repository:

- README.md
- a_maze_ing.py
- mazegen.tar.gz
- a configuration file
- all needed elements to build the maze generator package again Other files can be present. Also the fl
  Norm must pass on the python files, as well as mypy.

If any of the required elements is missing, the grade is 0.

Yes No

README.md file

Does the repository contain a README.md file at its root, and does it include all of the following?

- The first line is italicized and formatted exactly as: *This project has been created as part of the 42 curric by <login>, <login>.*
- A "Description" section explaining the project's purpose and providing a brief overview.
- An "Instructions" section with relevant details about compilation, installation, and/or execution.
- A "Resources" section listing external references (documentation, tutorials, etc.) and explaining how AI used, specifying for which tasks and which parts of the project.
- The complete description of the configuration file.
- The chosen maze algorithm.
- The reason of choosing this algorithm.
- A short documentation about the maze generator reusable module.
- The team and project management :
  - The roles of each team member.
  - Your anticipated planning and how it evolved until the end.
  - Wath worked well and what could be improved.
  - Have you used any specific tools? Which ones?

Other relevant sections / elements are allowed in this file.
If any of the required elements is missing, the grade is 0.

Yes No

Standard usage

Display

Run the a_maze_ing.py script, using a default configuration file.
You should get a maze randomly generated, displayed on the terminal or using a dedicated graphic window, and a way to interact with the program.

Yes No

Interactive menu

Among the possible interactions, control that the mandatory ones are present:

- Re-generate a new maze.
- A toggle button to show and hide the shortest path from entry to exit.
- A button to change the colours of the walls. Extra interactions are possible (e.g., changing the colour of '42' pattern, quitting the program, ...).

Control that the configuration files follow the requirements:

- Comments lines start with '#'.
- Each line is in the 'KEY=VALUE' format (lower case is also OK).
- The expected keys are present (WIDTH, HEIGHT, ENTRY, EXIT, OUTPUT_FILE, PERFECT).

✓ Yes ✕ No

Error Management

Edit the configuration file to ensure that the program correctly handles its errors:

- Remove a mandatory key.
- Add a line that does not respect the 'KEY=VALUE' format (e.g., without the '=' sign).
- Replace numbers by letters.
- Use an incorrect boolean parameter for 'PERFECT'.
- Use a wrong format of int tuple for 'ENTRY' or 'EXIT'. Reminder: in case of unexpected end of the progr evaluation stops and final grade is 0.

✓ Yes ✕ No

Output file

Format

Run the a_maze_ing.py script and verify that the first generated (and displayed) maze is also stored in the defined output file. You should find:

- 'HEIGHT' lines of 'WIDTH' chars each,
- an empty line,
- the 'ENTRY' tuple,
- the 'EXIT' tuple,
- the shortest path from entry to exit described using directions 'N', 'E', 'S', 'W'. Use the validation script provided with the subject to control the walls coherence. Control that the shortest path sequence in the file matches its visual representation. Any error will mark this question as failed.

✓ Yes ✕ No

Maze Generator module

Maze generation

The maze generator module follow the subject's expectations:

- A randomly generated maze, eventually using a suitable standard module of Python (e.g., the module 'random').
- Adopt a relevant behaviour in case some parameters may be inconsistent (e.g., the entry or exit cells a outside the maze, width or height is negative, ...).
- All cells can be reached, except, eventually, those creating the '42' pattern.
- Have walls all around the maze.
- There is no 3x3 or larger open zone without walls. Ask how this was implemented or verified in the cod
- There is the '42' pattern in the maze (except is size does not allow it, then you should have a message c terminal).
- A perfect maze if the 'PERFECT' key in the configuration file is set to True.
- The same maze is replicable when using an identical seed (update the configuration file or the code to Are all these expectations fulfilled?

- The same maze is replicable when using an identical seed (update the configuration file or the code to

Are all these expectations fulfilled?

Yes | No

Reusable module

Have the evaluated student/team re-generate the package, most preferably in a virtualenv or similar.
This must re-create the mazegen-*.tar.gz or .whl file.
Then, in a different virtualenv or similar, install the newly created package, and test it with the a_maze_ing.py and its
configuration file.
Is everything working properly?

Yes | No

Bonuses

Do you want more?

The bonuses can only be considered if all the previous questions were successful.
You can validate up to 5 different working bonuses. Each bonus must represent a fair amount of work.
Possible ideas:

- Support multiple maze generation algorithms.
- Add animation during maze generation.

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 | Norme | Cheat | Crast

Incomplete group | Concerning situation | Can't support / explain code

Conclusion

Leave a comment on this evaluation (2048 chars max)

Finish evaluation