Remember that the quality of the defenses, hence the quality of the of the school on the labor market depends on you. The remote defences during the Covid crisis allows more flexibility so you can progress into your curriculum, but also brings more risks of cheat, injustice, laziness, that will harm everyone's skills development. We do count on your maturity and wisdom during these remote defenses for the benefits of the entire community.

SCALE FOR PROJECT CUB3D

You should evaluate 1 student in this team

Git repository

`git@vogsphere-v2-bg.1:`

Introduction

Please respect the following rules:

- 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 segfault, no other unexpected, premature, uncontrolled or unexpected termination of the program, 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.

- You must also verify the absence of memory leaks. Any memory allocated on the heap must be properly freed before the end of execution.
You are allowed to use any of the different tools available on the computer, such as leaks, valgrind, or e_fence. In case of memory leaks, tick the appropriate flag.

Attachments

subject.pdf minilibx_opengl.tgz minilibx_mms_20200219_beta.tgz

Mandatory part

Executable name

Check that the project compiles well (without re-link) when you excute the `make` command and that the executable name is `cub3D`.

Yes No

Configuration file

Check that you can configure ALL the following elements in the configuration file. The formating has to be as described in the subject.

- the image resolution/window size - R
- north texture path - NO
- east texture path - EA
- south texture path - SO
- west texture path - WE
- sprite texture path - S
- floor color - F
- ceilling color - C
- the map (see subject for the map configuration details)

Also check that the program returns an error and exits properly when the configuration file is misconfigured (for example an unknow key, double keys, an invalid path..) or if the filename doesn't end with the `.cub` extension.

If not, the defence is over and the final grade will be 0.

Yes No

Technical elements of the display

We're going to evaluate the technical elements of the display. Run the program and execute the 5 following tests. If at least one fails, no points will be awarded for this section. Move to the next one.

- A windows must open at the launch of the program if the `--save` argument is not supplied. It must stay open during the whole execution and have the resolution as set in the configuration file.

- An image representing the inside of a maze must be displayed inside the window.

- When the argument `--save` is supplied, the program won't open a window but output a file in bmp format with the resolution as defined in the configuration file (R key).
Note that a subprocess opening up in the dock is not the same as a window.

- Hide all or part of the window either by using another window or by using the screen's borders, then minimize the windows and maximize it back. In all cases the content of the window must remain consistent.

- Set a resolution (R key) in the config file greater than the actual screen resolution. The program must resize the window to fit the screen resolution.

Yes No

User basic events

In this section we're going to evaluate the program's user generated events. Execute the 3 following tests. If at least one fails, this means that no points will be awarded for this section. Move to the next one.

- Click the red cross at the top left of the window. The window must close and the program must exit cleanly.

- Press the ESC key. The window must close and the program must exit cleanly. In the case of this test, we will accept that another key exits the program, for example Q.

- Press the four arrow keys (we'll accept WASD or ZQSD keys) in the order of your liking. Each keypress must render a visible result on the window, such as a player's movement/rotation.

Yes | No

Movements

In this section we'll evaluate the implementation of player's movement/orientation inside the maze.
Execute the 5 following tests. If at least one fails, this means that no points will be awarded for this section.
Move to the next one.

- The player's spawning orientation on the first image must be in accordance with the configuration file, test for each cardinal orientation (N, S, E, W).

- Press the left arrow (or A or Q) then the right arrow (or D). The player's view must rotate to the left then to the right as if the player's head was moving.

- Press the up arrow (or W or Z) then the down arrow (or S). The player's view must go forward and then backward in a straight line.

- Press A or Q (or the left arrow) then D (or the right arrow). The player's view must go to the left and then to the right in a straight line.

- During those four movements, was the display smooth? By smooth we mean is the game "playable" or is it slow.

Yes | No

Walls & Sprites

In this section we'll evaluate the walls and sprites implementation in the maze.
Execute the 5 following tests. If at least one fails, this means that no points will be awarded for this section.
Move to the next one.

- The walls texture vary depending on which compass point the wall is facing (north, south, east, west).
Check that the textures on the walls and perspective are clearly visible and correct.

- Check that if you modify the path of a wall texture or sprite image in the configuration file, it actually modifies the rendered texture or sprite when the program is re-executed.

- Also check that if you set a non-existant path it raises an error.

- Check that the sprite is displayed correctly and that it can be present multiple times in the same map.

- Check that the floor and ceilling colors are well handled when you modify them in the configutation file.

Yes | No

Error management

In this section, we'll evaluate the program's error management and reliability. Execute the 4 following tests. If at least one fails, this means that no points will be awarded for this section.
Move to the next one.

- Run the program using numerous arguments and random values. Even if the program doesn't require any arguments, it is critical that those arguments don't alternate or create unhandled errors.

- Check that there are no memory leaks. You can use the `top` or `leaks` command in another shell to monitor that the memory use is stable. The memory used must not increase each time an action is made.

- Roll either your arm or your face on the keyboard. The program must not show any strange behaviors and it must stay functional.

- Modify the map. The program must not show any strange behaviors and it must stay functional if the map is well configured, if not it must raise an error.

Yes | No

Bonus

We will look at your bonuses if and only if your mandatory part is excellent. This means that you must complete the mandatory part, beginning to end, and your error management must be flawless, even in cases of twisted or bad usage. So if the mandatory part didn't score all the point during this defence bonuses will be totally ignored.

When i'll be older i'll be John Carmack

Look at the subject bonus part and add one point for each bonus implemented and fully functionnal.

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

And more?

Same as before, but add one point when two more bonuses of the list are well implemented and fully functionnal. Round it up if necessary (9 bonuses is 5/5).

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 | No author file | Invalid compilation | Norme | Cheat

Crash | Leaks | Forbidden function

Conclusion

Leave a comment on this evaluation

make sure you handle all user input

Finish evaluation