SCALE FOR PROJECT NIBBLER (/PROJECTS/NIBBLER)

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 defense.

- 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

Subject (/uploads/document/document/1010/nibbler.en.pdf)

Preliminaries

Preliminaries

Check firstly the following points:

- There is something in the git repository.

- No cheating both student can explain their code.

- There can't be in the repository any source of
a library that wasn't written by the group itself.
It means that every external library used for the
project can't be in the repository. That includes
obviously graphical libraries, as well as utilitary
libraries such as Boost, etc. See the subject.

- The corrected group must supply a script or a group
of scripts allowing an easy and accurate installation
of dependencies according to the subject. The group
is therefore responsible for the setting up of the
environment on the system used for p2p correction
whatever state it is in.

- The corrector is the only judge in determing if that
phase is too long or no appropriate and can decide
unilaterally to end the p2p correction prematurely
by clicking the "incomplete work" flag. With great power

comes great reponsibility. Don't be a dick but don't
be naive either.

- In the main binary, the group CANNOT make ANY reference
to a specific graphic library (like OpenGL, SDL, etc).
That would be a pure and simple off-topic. This not only
includes display primitives but events primitives as
well. For example the presence of Qt or SDL types or
straight calls to functions of those librairies.

- The game loop must ABSOLUTELY be in the main program and
definitely not in the libraries. However, some libraries
impose their own loop, in this case, that loop MUST be
manipulated to be synchronized with the core loop and
be completely dependant of it. The opposite would be
off-topic and must be flagged "incomplete work". Check
the subject and below in case of doubt.

- A makefile containing the usual rules must compile the
binary as well as the 3 libraries dynamically.

- If the project includes less that 3 dynamic libraries
used during its execution use the "incomplete work"
flag.

If an element of this list isn't respected, the grading ends.
Use the appropriate flag. You're allowed to debate some more
about the project, but the grading will not be applied.

☑ Yes ☒ No

Feature testing

Launch

The main binary allows at least to choose the size of the
game board and select a dynamic library to display the game.
The game then runs normally.

☑ Yes ☒ No

Lauching errors

Lauching errors must be managed. The minimum is to manage the following:

- Insufficient number of arguments.
- Wrong height / width (negative numbers, too small, too big, no numbers ...)
- ...

It's not acceptable to just quit the program without giving any feedback.

☑ Yes ☒ No

Movements

The snake moves on its own, and it's possible to change the direction using the keyboard. This behavior is uniform in the 3 different libraries.

☑ Yes ☒ No

Nourriture

The snake can eat a "fruit", and grows from it. This behavior is uniform in the 3 different libraries since it's unrelated to the game logic.

☑ Yes ☒ No

Defeat

The game ends if the snake touches a wall or its own body. This behavior is uniform in the 3 different libraries since it's unrelated to the game logic.

☑ Yes ☒ No

Deliberate end of game

It is possible to deliberately end the game by pressing Esc or any other traditional way. This behavior is uniform in the 3 different libraries.

☑ Yes ☒ No

GUI update

It is possible to deliberately change the GUI by pressing
1, 2 or 3 according to the subject. This feature being
at the center of this project, make sure it is stable.
Check it at different moment of the execution, in any order,
by switching quickly from one to another, etc.

☑ Yes ☒ No

Implementation

We want to remind you that the game loop and therefore the game logic must be located in the main binary according to the requirement of the subject. NO exception allowed. If a library requires it own loop, that loop must be synchronysed with the main binary's loop. Yes it's tricky and that part of the code must be well designed. It especially implies that this graphic library was probably not adapted to this project. Freedom involves to make choices...

Consistant game loop

The game loop should look like this:

```
while (...)
{
eventManagement();
gameDataUpdate();
displayUpdate();
}
```

Is that the case?

☑ Yes ☒ No

Event management

Keyboard event management must be fully functional and
clean. An ifs forest is NOT acceptable here.

☑ Yes ☒ No

Core interface/Libraries

The aim of the project is double. Firstly, it is about breaking graphical aspects from program inputs by relocating them inside a graphical library. The project must offer the use of 3 different graphical libraries. Secondly, it is about interacting evenly with either one of the other graphical libraries created by the group.

The students must have designed a clever interface to interact with the display module. The minimum would be an interface looking roughly like the following:

```
class INibblerDisplay
{
public:
virtual void init(...) = 0;
virtual void getEvents(...) = 0;
virtual void updateGameData(...) = 0;
virtual void refreshDisplay(...) = 0;
virtual void stop(...) = 0;
};
```

We should find something to initialize the grapical library contained in the dynamic library, something to notify the library of the game to display, something to get inputs from the user, etc. According to the preliminaries, we would like to remind you that no reference of any sort to any of the libraries should exist in the main program, and therefore in this interface.

Is there an adapted interface in the main program which allows to communicated evenly with at least the 3 dynamic libraries encapsulating the 3 graphical libraries selected?

☑ Yes ☒ No

Object display plug_ins / Consistent design

The Nibbler's dynamic libraries can therefore be seen as plug-ins implementing a specific interface (meaning a "contract" exposing their different features available for the user's program). This implies that the binary's code as well as the libraries' must be in C++ according to the subject. However, communication between a program and libraries is in C. This implies the use of a specific syntactic construction "extern C" to advise the compilator that a small portion of the code is in C and not in C++.

These little C function are called the library's "entry points".

The entry points can return generic information on the library to ensure compatibility with the main program, but there MUST be an entry point allowing to recover an instance of the diplay class contained in the dynamic library in the form of a pointer to interface referred to in the last section. Otherwise the project misses the point.

Naive but acceptable example:

```
extern "C"
{
INibblerDisplay* getDisplayModule() { return new NibblerDisplayOpenGL(); }
}
```

The entry points must be very short functions that usually only returns data, such as a new instance. If you find out that a more complicated processing is done in the entry point, this section is not validated.

Check that the three libraries operate in the manner descibed above.

☑ Yes ☒ No

Dynamic library error management

Check out that the group arranged an error management when loading, using or unloading the dynamic libraries using exceptions. You must refuse error handling made without exceptions as well as scalar exceptions.

☑ Yes ☒ No

Le groupe

Group's project approach

Nibbler is a two man's project. Its relative complexity makes it an ideal candidate to start a proper division of labour approach.

- 5 stars -> Both students participated actively in the coding and the defense. They both know the overall functionning as well as implementation details from each other's part (the graphical library part for example).

- 4 stars -> Both students participated actively in the coding and know their respective part, but not in details the other part.

- 3 stars -> Not used here.

- 2 stars -> One of the students is behind on the project on the coding and during the defense, but visibly participated as much as he could.

- 1 star -> Not used here.

- 0 stars -> It's obvious than only one of the students did the job (whatever the reason is), whatever the quality of the project is.

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

Bonus

Sounds

There is one (or several) sound libraries, and they are manage the same way as the graphical libraries. Meaning via a dynamic library created by the students loaded at runtime.

☑ Yes ☒ No

Multiplayer

There is two or more snakes fighting for the food?!
(A single playe mode must still be possible).

☑ Yes ☒ No

Network game

The students made a bad-ass network multiplayer mode?
(Previous bonus is validated as well).

☑ Yes ☒ No

Additional bonuses

The students added additional features to the project improving the game experience? Count them here you can count up to 5 distincts ones.

Example of valid bonus features:
- A scoring system
- Bonus / Malus (fruits)
- A head easily distinguished from the rest of the body
- An increasing game speed
- Obstacles on the board
- etc ...

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 Forbidden function

Preview!!!

Conclusion

Leave a comment on this evaluation