C++ Project

Nibbler

Koalab koala@epitech.eu

Abstract: The purpose of this project is to create your own version of the game Snake, with at least 3 different GUIs. These GUIs being shared libraries.

Contents

I Snake

`Snake` is a classical game from the 70's. The simplicity of this addictive game resulted on its availability on most of existing game platform under a form or another. For the old young, `Snake` means hours and hours of slacking at school, using the mobile phone version such as the one of the `Nokia 3310`. For younger one, `Snake` is a Tech 2 project referring to an obscure game from prehistory ages, which interest only the old fossil pretending to be in the trend of their time.

It is never too late to discover this classical of video game: http://jeux-flash.jeu-gratuit.net/jeux_classiques/snake_250.html

As you can see, `Snake` is about moving a snake within a plan. This snake is represented by sections and it have to eat food, so that it can grow by a section each time it gets food. The game is over when the head of the snake hit an edge of the plan or one of the section. The goal of the game is to obtain a snake as long as possible.

Several versions of `Snake` exists. Some of them include obstacles, others have a score system, or bonuses etc.

II The project

`Nibbler` is a video game in which the player controls a snake moving within a plan. The window edges are the limits of the plan. If the snake hits one of these edges, or hits a part of his own body, the game is over.

II.1 Generalities

For your own culture it is very important to know several graphic libraries. This is why your final project must provides 3 different graphic libraries. The interest of this project being to make you handle dynamic libraries at the execution, the graphic rendering as well as output must be located within a dynamic library used for the execution. The body of your program must interact in a uniform way with one or another of your libraries.

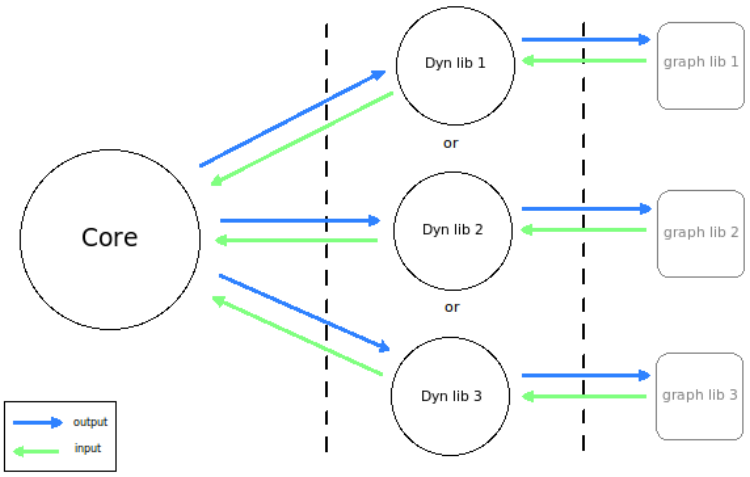

Figure 1: `Architecture`

Each GUI available for the program must be used as a shared library that will be loaded and used dynamically by the main program. It is strictly **FORBIDDEN** to do any reference to one or another graphic library that you want to use in your main program. Only your dynamic libraries can do it!

`Again: It is strictly FORBIDDEN to do any reference to one or another graphic library that you want to use in your main program.`

II.2 Dynamic libraries

You must use your dynamic libraries at the execution of your program. This means that you MUST use functions `dlopen`, `dlclose`, `dlsym` and `dlerror` to handle your dynamic libraries, as presented in class. Dependency to your libraries will not appears when passing your program as parameter to the command `ldd`.

Those libraries can be considered as plug-ins providing graphical interface capabilities for your main program. In NO CASE, they are influencing the game logic. They are used only to display the state of the game to the player and to obtain the player inputs for the main program.

You MUST NOT make any difference between one or another library of your main program. Each of your libraries MUST be handled in a generic and uniform manner. It is the generic aspect that interests your grader during the project defense.

Indices Class and material available on the e-learning, as well as the tutorial are important sources of information to optimize the realization of this part of the project. However, nothing replaces the thinking.

II.3 Graphic libraries

You can choose your 3 graphic libraries within the following list

- NCurses
- NDK++
- MinilibX
- Xlib
- GTK++
- OpenGL
- SFML ou SDL (Only one of them)
- Qt

If you want to use a graphic library that is not in this list, contact me to get an authorization. I recommend to contact me through the C++ forum on the intranet, so that your proposal can benefit your classmates too.

Some of these libraries cannot be installed on the standard dumps of the school. You should then consider that you will have to install manually these libraries, without privileged rights. You will acquire more than culture at this point.

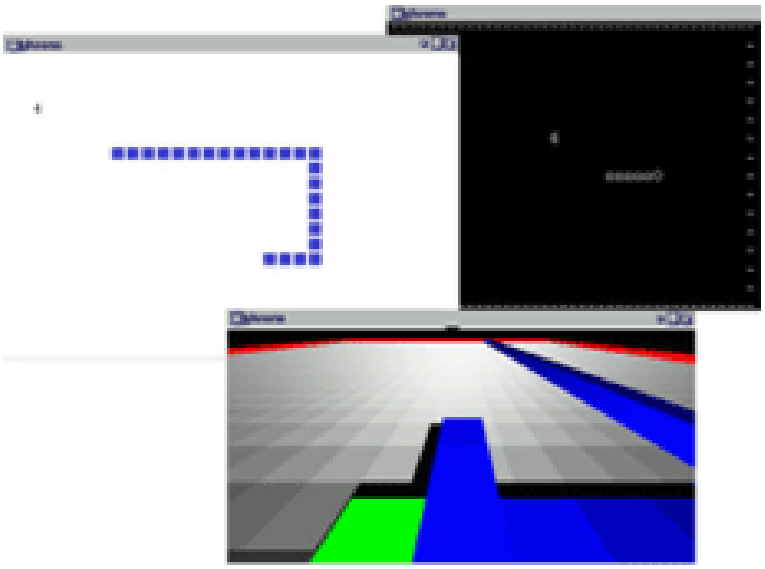

Figure 2: `Turn in examples`

II.4 The Game

Rules of the game are very basic and MUST be respected. These are the bases:

- The unit measure is the "case". The size of a case is up to you, but it MUST be reasonable and CAN vary with the graphic libraries you are using. A 1px case is too small while a 1000px is too large.
- The game area is a finite plan of cases. The edges of the plan can't be passed through.
- The snake starts with a size of 4 cases in the middle of the game area.
- Le snake progress forward automatically at a constant speed. Each section of its tail follows the exact same path that the head.
- The snake can turn to the right or to the left from 90° when the corresponding touch of the keyboard is pressed.
- The goal of this game is to feed your snake so that he can grow. The game area MUST NEVER have less that one element of food.
- One food element fill only one case of the area.
- When the head of the snake is over a case with food, the food disappear, and one section of one case is added at the tail of the snake. The added section appears in the first free case nearby the last case of the tail of the snake. If there is no free case, the game is over. If a new section of the serpent is added, then a new food element appears.

Once you are done with your project and the 3 dynamic libraries, you can look for a bonus by extending the game rules. These are some examples:

- A scoring system
- A bonus of food appears for a short period of time
- The head section looks different from the others sections
- The snake speed increases during the game
- The game area has some obstacles
- The size of the snake increases randomly when eating
- A speed boost limit when pressing the space bar
- ...

II.5 Usage

The results of your turn in compilation MUST BE a program and 3 dynamic libraries. The program name MUST be "nibbler" and the dynamic libraries MUST be named from the graphic library they are using. Only the executable name is imposed. However the libraries SHOULD have a name similar to "lib_nibbler_XXX.so", where "XXX" is the name of the graphic library used.

The exec "nibbler" must accept 3 parameters :

- Game area width
- Game area height
- The name of the dynamic library to use

Example :

```
1    >./nibbler 30 20 lib_nibbler_opengl.so
```

You MUST handle the following cases:

- If the number of argument passed to your program is different from 3, your program MUST display a usage and exit neatly.
- If the size of the area is impossible (Negative values, non numerical, too big, too small etc.) your program MUST display a relevant error message and then exit neatly.
- If the dynamic library does not exist or is not compatible, your program MUST display a relevant error message and exit neatly.

When your program is running, the 3 keyboard touch MUST behave in the following way:

- Left arrow: The snake turns of 90° to the left.
- Right arrow: The snake turns of 90° to the right.
- Esc : Game is over, closes the windows Terminates the program neatly.

III Instructions

You are more or less free to implement your program how you want it. However there are certain rules:

- The only functions of the `libc` that are authorized are those one that encapsulate system calls, and that don't have a C++ equivalent.
- Each value passed by copy instead of reference or by pointer must be justified. Otherwise, you'll loose points.
- Each value non `const` passed as parameter must be justified. Otherwise, you'll loose points.
- Each member function or method that does not modify the current instance and which is not `const` must be justified. Otherwise, you'll loose points.
- There are no `C++` norm. However if a code is reckoned to be unreadable or dirty, this code will be penalized. Be serious please!
- Keep an eye on this project instructions. It can change with time!
- I am very concern about the quality of my materials. Please, if you find out any spelling mistake, grammatical errors etc. please contact me at thor@epitech.net so that I can do a proper correction.

IV Turn in instructions

You must turn in you project in the repository provided by `Epitech`. Repository name is `cpp_nibbler`.

Repository will be cloned at the exact hour of the end of the project, intranet `Epitech` being the reference.

Only the code from your repository will be graded during the oral defense.

Good luck!