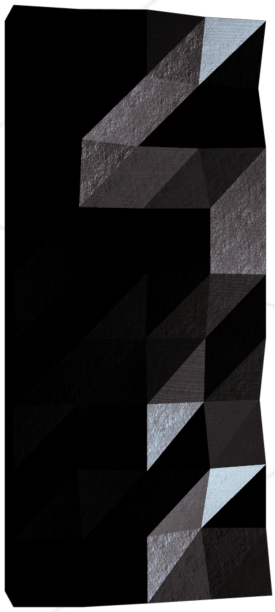

C++ Project

Nibbler

Summary: 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

Chapter I

Forewords

Between the Buried and Me is an American progressive metal band from Raleigh, North Carolina. They have released a total of six studio albums, as well as an EP, a cover album, and two live DVD/CDs. Their first album was released through Life Force records, and after that the majority of the group's releases were made through Victory Records, until their shift to Metal Blade in 2011 when they released their first EP, The Parallax: Hypersleep Dialogues through the label on April 12, 2011. Their most recent release, The Parallax II: Future Sequence was released on October 9, 2012.

Full article.

Chapter II

Snake

II.1 Introduction

Snake is a classical game from the 70's. The simplicity of this addictive game resulted on its availability on most of existing game platforms under a form or another. For "old" people, `Snake` means hours and hours of slacking at school, using the mobile phone version of the `Nokia 3310`. For youngsters, `Snake` is a `42` project referring to an obscure game from prehistoric ages.

II.2 Objectives

`Nibbler` is yet another snake video game written in `C++` including a major twist: dynamic libraries. Through this project, you will discover the power and usefullness of dynamic libraries used at runtime. Your goal is splitted in 4 parts : a main executable and 3 dynamic libraries that your main executable will load and use at runtime. Each library will embed everything your main executable might need to display the game and to get the player's inputs. The main executable will only focus on the game logic and interact with your dynamic libraries. The main executable must interact in a identical way with any of your libraries.

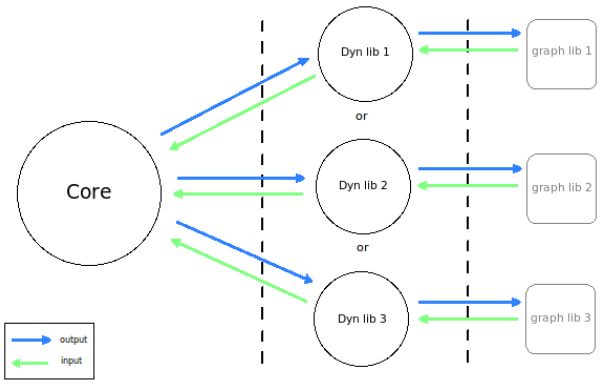

Figure II.1: `Architecture`

II.3 Generic instructions

- This project must be in `C++`.
- You are free to use any compiler you like.
- Your code must compile with: `-Wall -Wextra -Werror`.
- You are free to use any `C++` version you like.
- You are free to use any library you like.
- You must provide a Makefile with the usual rules.
- Any class that declares at least one attribute must be written in canonical form. Inheriting from a class that declares attributes does not count as declaring attributes.
- It's forbidden to implement any function in a header file, except for templates and the virtual destructor of a base class.
- The "keyword" `"using namespace"` is forbidden.
- Last but not least, you **MUST NOT** push any library that you did not write yourself on your repository ! Doing so equals zero at the defense. For instance, if one of your dynamic libraries uses `OpenGL`, you `MUST NOT` push `OpenGL` and/or its sources on your repository. Same for `Boost` or anything else. To ensure that you will be able to compile your work during the defense, you must provide a script or run a tool to set up your environment, and download and install your dependencies if needed. You should have a look at `CMake` for instance, or any other tool you like, or do that work yourself. Please read this part again, we won't accept complaints after a failed defense because you were not able to compile your work on a fresh session, whatever the version of the dump.

II.4 Mendatory part

II.4.1 Technical side

You must create a Snake game made of 4 parts : a main program and 3 dynamic libraries. Each dynamic library "embeds" a graphical interface and input handling routines for your game. Let's state this in another way: each available GUI and input handler for your game must only exist in a dynamic library that will be loaded and used dynamically by the main program at runtime. It is strictly **FORBIDDEN** to do any reference to any graphic library that you want to use in your main program. Only your dynamic libraries can do it ! Of course, nothing related to the game logic must appear in any of your dynamic libraries. This is the main executable's job !

`Again: It is strictly FORBIDDEN to do any reference to a graphic library that you want to use in your main program. Also, nothing related to the game logic must appear in any of your dynamic libraries.`

Your main executable must at least accept as parameters the game area's width and height. When run, your main executable uses one of your dynamic libraries to display the game. The choice can be random if you like, or always the same if you prefer. Anyway, once in game, it must be possible to switch from one dynamic library to one another by using `1`, `2` and `3`, thus changing the GUI.

Also, You must handle the following cases:

- If the number of argument passed to your program is wrong, your program must display a usage message 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. Too big and too small are up to you.
- If a dynamic library does not exist or is not compatible, your program must display a relevant error message and exit neatly.
- There must be a way to quit the game neatly. For instance by using the `esc` key or through a menu.

If you never use the functions `dlopen`, `dlclose`, `dlsym` and `dlerror` in your main executable, something is wrong. Your dynamic libraries can be considered as plug-ins providing graphical interface capabilities and input handlers for your main program. In no way whatsoever are they allowed to influence the game logic. They are used only to display the state of the game to the player and to obtain the player's input for the main program. You must not make any difference between your dynamic libraries in your main program. Each of your libraries must be handled in a generic and uniform manner. If one of the graphical libraries you chose can't fullfil this requirement, then this graphical library is obviously not suited for this project. Interfaces and abstractions are key here.

Also, you will soon discover that the ABI used to load and use dynamic libraries at runtime is `C`, not `C++`. As a consequence, you will need to use a special construct called `extern C`. This is **NOT** an excuse to use `C` instead of `C++` in your dynamic libraries. You will need `extern C` to create entry points, but apart from these entry points, you must handle class instances. This may seem obvious, but soon you'll discover that creating an instance from a class defined in a dynamic library and using that instance in you main executable is not trivial.

Just to be sure that everything is clear: keeping the logic out from your dynamic libraries implies that the game loop takes place in the main executable. If a graphical library forces its own loop, then this loop must be controled and synchronised from the main executable's loop. But to be honnest, this especially means that this specific graphical library is not suited for this project.

II.4.2 The game rules

- The unit of measure is the square. The size of a square is up to you, but it must be reasonable and can vary from one graphic library to another.
- The game area is a finite plane of squares. The edges of the plane can't be passed through.
- The snake starts with a size of 4 squares in the middle of the game area. The direction does not matter.
- The snake progresses forward automatically at a constant speed. That speed is up to you. Each section of its tail follows the exact same path than the head.
- The snake can't move backwards.
- The snake must be able to turn left or right using the keyboard.
- The goal of this game is to feed your snake to help it grow. The game area must never have less that one element of food. The food position can be random or pseudo-random if you want some control on where it spawns.
- One food element fills one and only one square of the area.
- When the head of the snake is over a square occupied by food, the food disappears, and one section is added to the tail of the snake on the next move on the same square as the previous last section.
- If the snake hits a wall or one of its own sections, the game is over.

II.5 Bonus part

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

- Make multiple game modes (changing speed, disappearing food, with a score, obstacles etc...)
- Sounds ! Please note that this bonus will be accepted if and only if the sound library is used the same way as your graphic libraries: through a dynamic library that you must create, load and use yourself at runtime. You don't need to create 3 though, but that would be badass.
- Multiplayer ! Two snakes fighting for the same food ! Wicked ! Please note that a single player mode must remain available.
- Multiplayer through network. Same as above. The project could hardly be more badass if you acheive that.

II.6 Turn-in and peer-evaluation

You must turn in the sources of your main executable and of your 3 dynamic libraries, and any script related to the setup of your environment. You must sort your sources in a relevant way. For instance a folder at the root of your repository for each one of the four parts and your makefile. Once compiled, the main executable and the 3 dynamic libraries should exist at the root of your repository or something similar.

Make sure that you never push anything that can be generated from something else on the repository. Pushing your dynamic libraries is as stupid as pushing an executable. Sprites and assets are Ok.

Please read again the section of the instructions about not pushing external libraries and/or their sources on your repository.

Enjoy !