Documentation Arcade

How can we make more libraries?

Hello, and welcome to the complete “doc” of our “cpp_arcade”. We will call it the Arcade for a better visibility.

To know what the Arcade is and how to use it, I suggest you first, to read the **README** file. Indeed, this documentation is not made to know about using the arcade but to know how to make more things with it!

So, we can say that this document concern **Developers**!

→ How to implement a library (like OpenGL, Qix, Pacman or others)?

To make a new library, first you can make the **Errors methods**!

Our Errors are based on *“Sources/Errors/Errors.hpp”* and *“Sources/Errors/Errors.cpp”*.

Every class: Arcade, Nibbler, NcursesLib… Has his equivalent in Errors!

That method allows you to know what Exceptions you have to throw and display a specific message for everything in all your classes.

The Error class inherit form **std::Exception** and it contains all the methods to throw Exceptions.

A **constructor** to set the message (_message) and the **stream** to display the message (cout, cerr…).

A **destructor** to delete it.

And finally, a “**what**” method. That returns the message!

When you throw your Error, it will display the message.

Per example:

```
if (!al_init())
        throw ErrorsAllegro(std::cerr, "Allegro init fail.");
```

➔ If the initialization of Allegro fail, it will print the message on the error output.

So, create your own ***Errors[yourLib].cpp*** and ***Errors[yourLib].hpp*** and make the same methods as the others, take example on the other files if you need.

Then, in you .cpp file, make an extern "C" method. The extern C makes a function in C++ have a C linkage.

Inside of that extern "C", make a **buildEngine**() function.

It's very important to name it like that, because the loader, will load it with that name!

Indeed our loader, use dlopen() to load games and graphics libraries with that contains buildEngine symbol!

With that buildEngine() function, you have to call the constructor of your library.

For a Graphic Library:

All of our public methods are located in: *"Sources/Lib_graph/IGfx.hpp"*

You have to implement **all of them** in your lib.

Every time you have to display, think about the clear function, indeed, it's very important to be sure that you display on a clear place. Think well about the order of you clear and prints!

- ➔ The **constructor** will take a name in parameter (the name of the window), a width and a height both of them for the size of the window.
- ➔ The **destructor** will erase, delete, destroy… all the textures you have to set, the window that you created, the fonts… Whatever you have loaded!
- ➔ **printLibSelection** is one of the first function to be called. You can print the Arcade Menu with it, that Arcade menu allows the user to choose between all the graphics libraries and all the games, when that user will press SPACE on the libraries he wants it will launch the game with the chosen libs!

- ➔ The **addAsset** method allows you to fill a map of Assets, the assets allow you to store all of your assets in a single map. For some libraries you should probably fill your map by loading textures in it (like in SDL or Allegro).
  That will optimize your program because all of the assets will be loaded once!

- ➔ The **printAsset** method takes the name of the asset to print, his position in x and in y. That method will be called in your games when you want to display an asset.
  If you can't find the asset, you should draw the name contained in you Asset. That will provide you from printing nothing if a font is not find!

- ➔ The **clearAsset** allows you to clear your map of assets. You'll have to call it every time you change games! Indeed, if you don't clear, when you change your game some assets of the previous games will persist.
- ➔ **GetLibName** returns the name of the lib: _libName.
- ➔ **GetWidth** returns _width.
- ➔ **GetHeight** returns _height.
- ➔ **printTitle** allows you to print the title of the game! You should probably use **printAsset** in it, if you want to display an image. Don't forget to **printAsset** the default name if you can't find the font.
- ➔ **printScore** allows you to print the score of the current player.
- ➔ **printHighScore** allows you to print all the highScores of the current game.
- ➔ **printGameMap** is provided to print the map of the game.
- ➔ **printGameMenu** Is made to display the pause menu. When the user press SPACE on the game it will display it, and there will be different options like resume, return menu and quit, if the users select one of them and press SPACE, it will select that option.
- ➔ **clearGameMenu** is provided to clear that pause menu. It will be useful because it allows you too clear only the pause menu and not all the screen. That can be nice to print the pause menu in front of the game, and not clear it.
- ➔ **printGameOptionMenu** prints the options menu of the pause menu. Indeed, if you want to have some options on your game, you can use that function to display them.
- ➔ **printGameOver** is provided to print the gameOver menu, if the user loses the game, it will display that screen until he presses SPACE to relaunch the game or ESCAPE or Q to quit.
- ➔ **printIntro** prints the intro of the game, when it's displayed, the user can press SPACE to launch the game.
- ➔ **Display** is obviously the function that allows you to display, indeed you will call it to display all of the Assets that you have set, per example in Allegro, that function allows you to call al_flip_display: that function display all of your images, text… newly set.
- ➔ **Clear** is the method to clear the screen. That is very useful when you have to draw because it clears all the screen. That function is often call in the core of the arcade.

➔ **getEvent** is one of the most important method in all the libs. That method is there to catch all the keyboard inputs we want:

- ESCAPE: back to menu or quit when you are on the menu.
- SPACE: to confirm or pause on game
- UP ARROW: go up
- DOWN ARROW: go down
- LEFT ARROW: go left
- RIGHT ARROW: go right
- Q to quit
- A previous game
- Z next game
- E previous graphic library
- R next graphic library

That function is called every time we need a keyboard input.

For a Game Library:

First, I suggest you to find all of the assets of your games, and create a folder in the folder Resources: *"Resources/[YOUR_GAME]".* Indeed, these assets will be your assets, for your game. Choose them well!

When that folder is created, create a folder **1D, 2D, Data and Score.**

In **the 1D folder**, you'll have to put all the assets concerning terminal libs, you can't load texture with them, but you can per example make ASCII ART to be the best in terminal libraries! So, your assets will be ".txt" file with you text in it.

In **the 2D folder**, you'll put your assets concerning 2D libraries like SDL, Allegro, SFML… You should probably use ".png" images to have the transparency.

In **the Data folder**, you have to create your map, ***composed with digits (0 to 9)***, every digit matches a specific asset in the map. The map has to be called *level[NUMBER].txt*.

Finally, in **the score folder**, you'll have to create a hidden file *".highscore".* Containing all the best scores for a specific game.

All of our public methods are located in: *"Sources/Games/IGame.hpp"*

You have to implement **all of them** in your lib.

Every time you have to display, think about the clear function, indeed, it's very important to be sure that you display on a clear place. Think well about the order of you clear and prints!

In your ".cpp" file, create all the **statics Assets** that contains all the properties of the Assets you want: **name, path1D, path2D, color, width, height**.

All of these properties allow you to draw and display your assets.

This has to be built like that:

```
static Asset MyAsset {
        .name = MYASSET,
        .path1D = " Resources/[GAME]/1D/[GAME].png",
        .path2D = "Resources/[GAME]/2D/[GAME].png",
        .color = {
            .r = [VALUE],
            .g = [VALUE],
            .b = [VALUE],
            .a = [VALUE],
            .pair = [PAIR]
        },
        .width = [VALUE],
        .height = [VALUE]
}
```

- ➔ **getGameName** returns the _gameName.
- ➔ **Launch** is the main function of your game. It loops until the game is finished, you should cut launch into several functions like: loopIntro, loopPause. Take a look in our games! It should give you some ideas.
  Think about calling all your display functions like: **display, printTitle, gameMap, High Score**. It's very important to call them there, and in a good order. That function is called in the arcade.
  It's also the function that returns to the Arcade the keys that are pressed in the game. Indeed, that function should handle if the user press Space, Q or Escape per example.
- ➔ At the beginning of **launch** you have to load all of your assets, create a function initGame, and addAsset all of your assets in it. It will fill your map of assets!
  We have also created _vector_pause, it's used to store the opitons in the Pause Menu. You can fill it too.
- ➔ **setGfx** is created because it allows the game to have access to the graphical library provided by the Arcade.

- ➔ For every game that you make, you have to load a Map! Create a function **loadMap**() and copy-paste one of the other **loadMap**() in the repository. The only thing to change is the numbers of the assets that you want to display on the map.

Obviously, you can implement many more functions if you want! But be careful they mustn't change the operation of the program. These functions must work just for your libs, and we don't have to modify the IGfx or IGame interfaces to make it work.

*******NB****:***

At the root of the repository, there is a Makefile that compile all of the project, don't forget to modify it to compile all of the project with your new library!