Arcade Documentation

Maya HILL, Nathan GUIU, Mathis LORENZO

Implement a new Graphic librairy

In the folder "*src/Librairies*", you can add a new folder with the name of your librairy (ex: *Vulkan*)
In this new folder, you need to add 3 files:

- **Makefile**: Used to compile your librairy in "*.so*" file

- ***[LibName]*Module.hpp**: This file contains all functions that you will code for this librairy. You need to create a class "*[LibName]*Module" that will inherit from "*IDisplayModule*". You must at least implement all functions present in the class "*IDisplayModule*". You also need to add a "*std::map<int, Arcade::KeyState>*" object called "*mapEvent*". This object must contains as first element a keyboard key as defined in the labrairy. (ex: SFML: key a = "*sf::Keyboard::A*") and as second element the value we want to get from the "*Arcade::Keystate*" enum.
Be sure to add all this elements in the namespace "***Libs***"

- ***[LibName]*Module.cpp**: This file contains all implementations of the functions present in the "*.hpp*" file.
This file must contains a function like this:

```
extern "C" {
    std::shared_ptr<Libs::IDisplayModule> create(void)
    {
        return std::make_shared<Libs::[LibName]Module>();
    }
}
```

This function is used to load the library.

[*LibName*]Module.cpp functions:

void Libs::[*LibName*]Module::init(std::string name, int w, int h)

This function take a name for the window, a width and a height as parameters. These arguments are used to create the window. In this function, you need to put everything you need to init a window with your librairy.

void Libs::[*LibName*]Module::my_clear() const:

This function is used to clear the window before drawing all elements composing the window.

void Libs::[*LibName*]Module::display() const:

It's the last function called when the widow is refreshed. This function update the window with the new elements.

void Libs::[*LibName*]Module::close():

This is the function called we the graphic librairy is changed. This goal of this function is to free all memory used by the librairy.

bool Libs::[*LibName*]Module::isOpen() const:

This function simply return whether the current window is open or not.

void Libs::[*LibName*]Module::setBackgroundColor(int r, int g, int b):

This function is used to change the color of the window's background.

**void Libs::[*LibName*]Module::draw(
std::unordered_map<std::string, Arcade::Rect_t> shapes,
std::unordered_map<std::string, Arcade::Sprites_t> sprites,
std::unordered_map<std::string, Arcade::Text_t> texts)**

You need to write inside this function, everything you need to draw all elements in the window. This function is used every time the windeow is refreshed. It takes as arguments every assets we need to draw on the window.

**std::vector<Arcade::KeyEvent_t>
Libs::[*LibName*]Module::getEvents():**

This function get all events from the keyboard or mouse, analyse them and return an "*std::vector<Arcade::KeyEvent_t>*" to send the informations through the core and the game.
This function also close the window if needed.

void Libs::[*LibName*]Module::setSizePixel(std::pair<int, int> size):

This function receive a "*std::pair<int, int> size*" as argument which contains the width and the heigth of a case in the window in pixels.

void Libs::[*LibName*]Module::clearAssets():

This functions clear all lists of objects use by the lib. We use this function when we change the graphic librairy.

void Libs::SFMLModule::setSizeWindows(std::pair<int, int> size):

This functions receive a "*std::pair<int, int>*" containing the width and the height we want to set to the window.

Implement a new Game librairy

In the folder "*src/Games*", you can add a new folder with the name of your game (ex: *Centipede*)
In this new folder, you need to add 3 files:

- **Makefile**: Used to compile your game in "*.so*" file

- ***[GameName]*Module.hpp**: This file contains all functions that you will code for this librairy. You need to create a class "*[GameName]*Module" that will inherit from "*GameModule*". You must at least implement all functions present in the class "*GameModule*".
Be sure to add all this elements in the namespace "***Games***"

- ***[GameName]*Module.cpp**: This file contains all implementations of the functions present in the "*.hpp*" file.
This file must contains a function like this:

```
extern "C" {
    std::shared_ptr<Games::IGameModule> create(void)
    {
        return std::make_shared<Games::[LibName]Module>();
    }
}
```

This function is used to load the librairy.
You can put your assets in a "*assets/[GameName]*" folder.
All your assets must have the extension ".png".
For each assets ".png", you must provide a asset with the same name but with the ".txt" extension. These files are used for the terminale librairies.

[*GameName*]Module.cpp functions:

Games::[*GameName*]Module::[*GameName*]Module():

The constructor is used to init any assets we need like a map or sprite.

void Games::[*GameName*]Module::restartGame():

We call this function when we need to restart the game.

void Games::[*GameName*]Module::handleEvents(std::vector<Arcade::KeyEvent_t> event):

This function handle all events for the game. It receive a "*std::vector<Arcade::KeyEvent_t>*" containing all keyboard and mouse inputs received from the lib.

std::unordered_map<std::string, Arcade::Rect_t> Games::[*GameName*]Module::getShapes():

Returns a "*std::unordered_map<std::string, Arcade::Rect_t>*" containing all shapes of the game.

std::unordered_map<std::string, Arcade::Sprites_t> Games::[*GameName*]Module::getSprites()

Returns a "*std::unordered_map<std::string, Arcade::Sprites_t>*" containing all sprites of the game.

std::unordered_map<std::string, Arcade::Text_t> Games::[*GameName*]Module::getTexts()

Returns a "*std::unordered_map<std::string, Arcade::Text_t>*" containing all texts of the game.

std::pair<int, int> Games::[*GameName*]Module::getSizeWindow():

Returns a "*std::pair<int, int>*" containing the width and the height of the window.

std::pair<int, int> Games::[*GameName*]Module::getSizePixel():

Returns a "*std::pair<int, int>*" containing the width and the height of a case in the window.

void Games::[*GameName*]Module::update():

You need to put everything you need to update the current state of the current game in this function.

void Games::[*GameName*]Module::restartGame():

This function is called when we need to restart the game when the current game is finished.

void Games::[*GameName*]Module::destroyGame():

This function is called when we need to destroy every assets of the game a free all the memory used by it.

void Games::[*GameName*]Module::createMap():

This function is used to initiate and create the map for the current game.