Implement new libraries

The `IGraphic` interface is used as a base for all our graphic libraries. It is imperative to use it if you want to implement a new graphic library.

Graphic libraries that you can implement:

The arcade is currently compatible with a limited amount of libraries.
Here is a list of currently compatible libraries :

- NDK++: `arcade_ndk++.so`
- aa-lib: `arcade_aalib.so`
- libcaca: `arcade_libcaca.so`
- Allegro5: `arcade_allegro5.so`
- Xlib: `arcade_xlib.so`
- GTK+: `arcade_gtk+.so`
- SFML: `arcade_sfml.so`
- Irrlicht: `arcade_irrlicht.so`
- OpenGL: `arcade_opengl.so`
- Vulkan: `arcade_vulkan.so`
- Qt5: `arcade_qt5.so`

However, you can add your own libraries as long as they are correctly implemented. Moreover, when your dynamic library is created, you must place it in the `./lib` folder for it to be used by the program. If you don't, it won't be used by our arcade.

HOW TO : add a new graphic library

All the graphic libraries loaded in the program must be dynamic.

Here is an example of implementation for the Ncurses library, that you can use as a template for your own implementation :

Filename: NcursesGraphicLib.cpp

```
extern "C"
{
    std::unique_ptr<IGraphic> entryPoint(void)
    {
        return (std::make_unique<NcursesGraphic>());
    }
}
```

You will also need to create a map to associate the key with the key detection of the library you want to implement:

Filename NcursesEventKey.hpp

```cpp
#ifndef NCURSESEVENTKEY_HPP_
#define NCURSESEVENTKEY_HPP_
#include "IGraphic.hpp"
#include <curses.h>
#include <unordered_map>

std::unordered_map<char, eventKey> keyEvent = { { 'a', eventKey::A },
    { 'b', eventKey::B }, { 'c', eventKey::C }, { 'd', eventKey::D },
    { 'e', eventKey::E }, { 'f', eventKey::F }, { 'g', eventKey::G },
    { 'h', eventKey::H }, { 'i', eventKey::I }, { 'j', eventKey::J },
    { 'k', eventKey::K }, { 'l', eventKey::L }, { 'm', eventKey::M },
    { 'n', eventKey::N }, { 'o', eventKey::O }, { 'p', eventKey::P },
    { 'q', eventKey::Q }, { 'r', eventKey::R }, { 's', eventKey::S },
    { 't', eventKey::T }, { 'u', eventKey::U }, { 'v', eventKey::V },
    { 'w', eventKey::W }, { 'x', eventKey::X }, { 'y', eventKey::Y },
    { 'z', eventKey::Z }, { KEY_F0, eventKey::SPACE }, { '\n', eventKey::ENTER },
    { KEY_LEFT, eventKey::LARROW }, { KEY_RIGHT, eventKey::RARROW },
    { KEY_DOWN, eventKey::BARROW }, { KEY_UP, eventKey::UARROW },
    { KEY_BACKSPACE, eventKey::DELETE }, { KEY_STAB, eventKey::TAB } };

#endif /* !NCURSESEVENTKEY_HPP_ */
```

General methods:

`void createWindow(std::string title, int width, int height)`

- Creates a graphical window with the specified title and dimensions.

`void clearWindow()`

- Clears the graphical window.

`void destroyWindow()`

- Destroys the graphical window.

`bool isOpenWindow()`

- Returns a boolean indicating whether the graphical window is open or closed.

`eventKey getEvent()`

- Retrieves the latest event from the graphical window, such as a keyboard press or mouse click.

Display methods

`void displayText(const text& text)`

- Displays text on the graphical window. The text parameter contains information such as the position, size, font, and color of the text to be displayed.

```
void displayShape(const shape& shape)
```

- Displays a shape on the graphical window. The shape parameter contains information such as the position, size, color, and shape type of the shape to be displayed.

```
void displaySprite(const sprite& sprite)
```

- Displays a sprite on the graphical window. The sprite parameter contains information such as the position, size, texture, and color of the sprite to be displayed.

```
void displayWindow()
```

- Displays the graphical window.