Documention Arcade

How to use and append libraries

Hello and welcome to the document of the arcade project.

⚠ **This document explains how to create and add new libraries. If you want to know how the project works, I suggest you read this README.**

Graphic Library

All of our methods are located in: **"Graphs/include/[libname].hpp"**

All of them must be implemented in your library.

FIY: When you want to display something, don't forget to call the clear function. It is important to avoid issues. Also, think about the order or your calls functions

The Constructor:

Will initialize the library and setup the type and name of the it.
It takes no parameter.

ex:

```
Graph::SDL2 *SDL2 = new Graph::SDL2();
```

The Destructor:

Won't do anything.

Open:

Will plays the role of the constructor. Create the window.
It takes the title of the window and his icon.

ex:

```
void open(std::string const &title = "", std::string const &icon = "");
```

Close:

Will plays the role of the deconstructor. Will erase, delete, destroy the window and all memory allocated.
It takes no parameter.

ex:

```
void close() const;
```

SetTitle:

Will changes the title of the window.
It takes the title as only parameter.

ex:

```
void setTitle(std::string const &title) noexcept;
```

SetIcon:

Will changes the icon of the window.
It takes the path to the image as only parameter.

ex:

```
void setIcon(std::string const &icon);
```

GetEvent:

Will retrieves the current event.
It takes no parameter.

ex:

```
Arcade::Events getEvent() noexcept;
```

GetEventChar:

Will retrieves the letter key of the last event.
It takes no parameter.

ex:

```
char getEventChar() const noexcept;
```

CheckCollision:

Will checks if there is a collision between two entities.
It takes as parameters: - Position of the first entity. - Size of the first entity. - Position of the second entity. - Size of the second entity.

ex:

```
int checkCollision(Position const &pos1, Size const &size1, Position const &pos2, Size const &size2) const noexcept;
```

DisplayWindow:

Displays the window after clearing it.
It takes no parameter;

ex:

```
void displayWindow() noexcept;
```

DisplayImage:

Will displays an image (= pixel array) according to its components.
It takes as parameters: - Position of the image. - Form of the image. - The color of the image. - The Size of the image.

ex:

```
void displayImage(Graph::Position &pos, Graph::Form &form, Arcade::ColorIdx &idx, Graph::Size &size);
```

DrawRect:

Will draws a rectangle.
It takes as parameters: Position of the rect. Color of the rect. Size of the rect.

ex:

```
void drawRect(Graph::Position &pos, Arcade::ColorIdx &idx, Graph::Size &size) noexcept;
```

Game Library

First of all, you need to have all of your game assets. Therefore, create a folder in: **"Ressources/Games/[Game_name]"**

You have to put your forms assets (characters, walls, items) in: **"Ressources/Games/[Game_name]/forms/"**

FIY: When you want to display something, don't forget to call the clear function. It is important to avoid issues. Also, think about the order or your calls functions

GetType

Will gets the type of the library.
It takes no parameter.

ex:

```
Arcade::Type getType() const noexcept = 0;
```

GetName

Will loads a scene and create all the components.
It takes the requested library as parameter.

ex:

```
void load(Graph::AGraph *lib) = 0;
```

HandleEvents

Will reacts to the events related on the elapsed time and simulate the game.
It takes the elapsed time as parameter.

ex:

```
void handleUpdate(double elapsedTime) const noexcept = 0;
```

Display

Will displays all the components.
It takes no parameter.

ex:

```
void display() const noexcept = 0;
```

FIY

You can implement as many functions as you want, but keep in mind, these functions **won't** work with the others libraries.