Arcade Doc

Module

```
namespace arcade
{
    class Module;
}

class Module
{
    protected:
    struct {
            size_t w;
            size_t h;
            uint16_t wUnit;
            uint16_t hUnit;
            uint16_t getWidthInUnit()
            {
                return (uint16_t) wwUnit;
            }
            uint16_t getHeightInUnit()
            {
                return (uint16_t) hhUnit;
            }
    } _size;
    sf::RenderWindow _sfWindow;
    SDL_Surface _sdlWindow;
    WINDOW _ncWindow;
    public:
    virtual void initWindow(const size_t &width = 800, const size_t &height = 600, const
std::string &title = "Arcade Game") = 0;
    virtual void destroyWindow() = 0;
    virtual void setObject(std::map<std::string, Object> &gameDatas, const std::string
&name, const std::string &sfTexturePath, const std::string &sdlTexturePath, const
std::string &ncTexturePath, const int &x = 0, const int &y = 0);
    virtual void setObjectCoordinates(std::map<std::string, Object> &gameDatas, const
std::string &name, const int &x, const int &y);
    virtual void setObjectTexture(std::map<std::string, Object> &gamesData, const
std::string &name, const std::string &sfTexturePath, const std::string &sdlTexturePath,
const std::string &ncTexturePath);
    virtual void destroyObject(std::map<std::string, Object> &gameDatas, const
std::string &name);
    virtual void render(const std::map<std::string, Object> &gamesData) = 0;
    virtual int getInputs() = 0;
}
```

class Module:
Module is an interface between the games and the libraries.
Contains all the lib functions.

Each game has a unique source code for each graphic library.
Module defines the methods that will be used by each game. All Module child
class will override this method for each graphic library (SFMLModule, NCurseModule...).

virtual void initWindow(const size_t &width = 800, const size_t &height = 600, const std::string &title = "Arcade Game") = 0;

Create a window of type sf::RenderWindow or SDL_Surface or WINDOW depending on the graphical lib used.

Parameters:
const size_t width Width of the window in cols and not in pixels.
const size_t height Height of the window in rows and not in pixels.
const std::string title Title of the window.

virtual void destroyWindow() = 0;

Destroy the current window.

virtual void setObject(std::map<std::string, Object> &gameDatas, const std::string &name, const std::string &sfTexturePath, const std::string &sdlTexturePath, const std::string &ncTexturePath, const int &x = 0, const int &y = 0);

Initialize an object and add it to the game data.

Parameters:
std::map<std::string, Object> gameDatas Current game data.
const std::string name Name of the object.
const std::string sfTexturePath Path to the picture file storing the object texture.
Only displayed for others lib than ncurse.
const std::string sdlTexturePath Path to the SDL texture file (must be .bmp file).
const std::string ncTexturePath Path to the Ncurse module texture file.
const int x X coordinate of the object in cols and not in pixels.
const int y Y coordinate of the object in rows and not in pixels

virtual void setObjectCoordinates(std::map<std::string, Object> &gameDatas, const std::string &name, const int &x, const int &y);

Set the coordinates of a given object into game data.

Parameters:
Std::map<std::string, Object> gameDatas Current game data.
const std::string name Name of the object.
const int x X coordinate of the object in cols and not in pixels.
const int y Y coordinate of the object in rows and not in pixels.

virtual void setObjectTexture(std::map<std::string, Object> &gamesData, const std::string &name, const std::string &sfTexturePath, const std::string &sdlTexturePath, const std::string &ncTexturePath);

Set the texture of a given object into game data.

Parameters:
std::map<std::string, Object> gameDatas Current game data.
const std::string name Name of the object.
const std::string sfTexturePath Path to the picture file storing the object texture.
Only displayed for others lib than ncurse.
const std::string sdlTexturePath Path to the SDL texture file (must be .bmp file).
const std::string ncTexturePath Path to the Ncurse module texture file.

virtual void destroyObject(std::map<std::string, Object> &gameDatas, const std::string &name);

Destroy an object from its name.

std::map<std::string, Object> gameDatas Current game data.
const std::string name Name of the object to destroy.

virtual void render(const std::map<std::string, Object> &gamesData) = 0;

Render all the object of the current game data.

Parameters:
std::map<std::string, Object> gameDatas Current game data.

virtual int getInputs() = 0;

Get the value of the current pressed key.

Returns:
Int value of the current pressed key

Game

```
namespace arcade
{
    class Game;
}

class Game
{
    protected:
    std::string _name;
    Module _module;
    std::map<std::string, Object> _data;

    public:
    virtual ~Game() = 0;
    const std::string &getName() const;
    Module getModule() const;
    const std::map<std::string, Object> &getData() const;
    void setName(const std::string &name);
    void setModule(Module module);
    void setData(const std::map<std::string, Object> &data);
    virtual void launchGame() = 0;
    virtual void coreGame() = 0;
};
```

Class Game:
Game is an interface for each game source code.

Game methods launchGame and coreGame() will be overwritten by its child class because each game has a different source code.
Each game will use methods from _module field.

virtual void launchGame() = 0;

Initialize the _data map then call coreGame().

virtual void coreGame() = 0;

Loops the game.