Arcade Guide

Introduction

In this guide you will learn how to implement a new game or graphic library to the arcade module. This project was conducted during a period of 5 weeks with Ewan Sioux, Adrian Lalardie, and Eliott Jouan.

Requirements

Before starting, it is primordial you have the following dependencies on your machine:

- SDL2
- SDL_image
- SDL_mixer
- SDL_ttf
- Ncurses
- SFML

Write a new game

Structure

Every game you make must be compiled as a **shared library** and be put inside the "lib/" folder so the Core of the arcade can find the library.

A game must inherit from the `IGameLib` interface and have it as public.

The `IGameLib` is defined as the following. It contains pure virtual methods that **HAVE** to be implemented.

```
namespace Arcade {
    // Generic interface used for the implementation of games librairies.
    class IGameLib : public ILib {
        public:
            // Destroy the IGameLib object, closes all associated resources.
            virtual ~IGameLib() = default;

             // Get the map of the current game instance.
            virtual std::vector<std::string> getGrid() = 0;

            // Get the score.
            virtual int getScore() = 0;

            // Start the instance of the game.
            virtual void start() = 0;

            // Store the key given as parameter in the instancied game.
            virtual void sendInput(Arcade::Keys key) = 0;

            // Get the Color Map object. Used to define the color of each associated character.
            virtual std::unordered_map<char, Arcade::Colors> getColorMap() = 0;

            // Get the Char Display Map object. Used to define replacement display characters.
            virtual std::unordered_map<char, char> getCharDisplayMap() = 0;

            // Get the Sprites Map object. Used to define sprite replacements for characters.
            virtual std::unordered_map<char, std::string> getSpritesMap() = 0;

            // Get the Game Music sound path
            virtual std::string getGameMusic() = 0;

            // Get the In Loop Music sound path. Played each frame
            virtual std::string getInLoopMusic() = 0;

            // Set the _map of the instancied game to the map given as parameter.
            virtual void setMap(std::vector<std::string> map) = 0;
    };
}
```

IMPORTANT

your CPP implementation of the game you want to create must contain a C defined "entryPoint".

For example:

```
extern "C" Arcade::IGameLib *entryPoint()
{
    return new Arcade::Snake();
}
```

This will be used by our shared library loader (DLLoader) that will load the .so library and load the entryPoint as a C object.

Write a new graphic library

Structure

Every graphic library you make must be compiled as a **shared library** and be put inside the "lib/" folder so the Core of the arcade can find the library.

A graohic library must inherit from the `IGraphicLib` interface and have it as public.

The `IGraphicLib` is defined as the following. It contains pure virtual methods that **HAVE** to be implemented.

```
namespace Arcade {
    // Generic interface used for the implementation of graphic librairies.
    class IGraphicLib: public ILib {
        public:
            // Destroy the IGraphicLib object, closes all associated resources.
            virtual ~IGraphicLib() = default;

            // display the map. A map is represented by a vector of strings, these contain the characters to display that will then b
            virtual void displayGrid(std::vector<std::string> map) = 0;

            // Get the user input from the graphical library
            virtual Arcade::Keys getUserInput() = 0;

            // start the graphic library, inside this function are initialized all of the components necessary for the game display.
            virtual void start() = 0;

            /*
             * Set the colors & display characters that will then be used on display.
             *
             * @param colorMap: unordered_map of characters & color items represented by Arcade::Colors that will set the color to u
             * @param charMap: unordered_map of characters. Maps a character to a display character. Usefull for Ncurses if you want
             */
            virtual void setGraphicMaps(std::unordered_map<char, Arcade::Colors> colorMap, std::unordered_map<char, char> charMap) =

            /**
             * Set the sprites to use on display
             *
             * @param spriteMap: unordered_map of characters to strings that represent the path of the asset to use as sprite. this r
             */
            virtual void setSpritesMap(std::unordered_map<char, std::string> spriteMap) = 0;

            // Display score on screen
            // @param score: score of player
            // @param grid: Map that is used to calculate the position of the score display
            virtual void displayScore(int score, std::vector<std::string> grid) = 0;

            // Plays a sound. Used to play background music & ephemere sounds.
            // @return std::size_t: id of the sound that can then be passed to "stopSound" to stop that particular sound
            virtual std::size_t playSound(std::string soundPath) = 0;

            // Stops a sound by its ID
            // @param soundId: sound ID that was generated by "playSound" to stop said sound.
            virtual void stopSound(std::size_t soundId) = 0;

            // display high score of player on screen
            // @param score: high score of player
            // @param grid: Map that is used to calculate the position of the high score display
            virtual void displayHighScore(int score, std::vector<std::string> grid) = 0;
    };
}
```

IMPORTANT

your CPP implementation of the graphic library you want to create must contain a C defined "entryPoint".

For example:

```
extern "C" Arcade::IGraphicLib *entryPoint()
{
    return new Arcade::Sfml();
}
```

This will be used by our shared library loader (DLLoader) that will load the .so library and load the entryPoint as a C object.