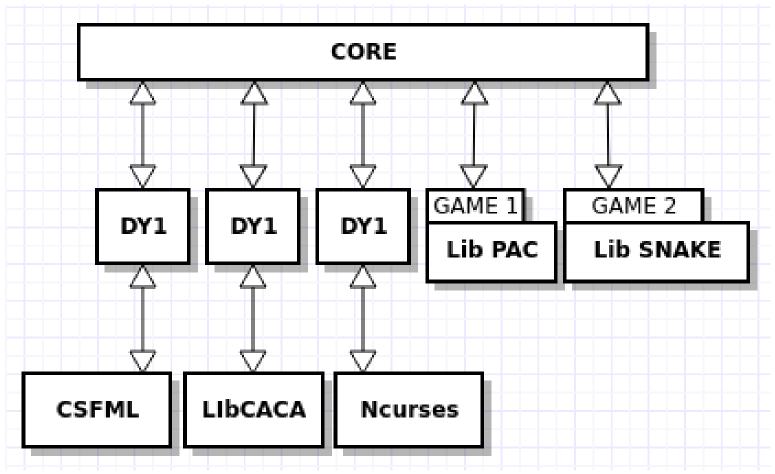
ARC.hpp, standards for the CPP_Arcade project

Arcade is a **gaming platform**: a program that lets the user choose a game to play and keeps a register of player scores. To be able to deal with the elements of the gaming plateform at run-time, graphics libraries and games must be implemented as **dynamic libraries**, loaded at runtime. Each GUI available for the program must be used as a shared library that will be loaded and used dynamically by the main program.

Project Architecture



List of common standards use across the Cpp_Arcade Project:

Interaction

```
/* Controls Interactions */

enum Interaction {
        MOVE_UP,
        MOVE_DOWN,
        MOVE_LEFT,
        MOVE_RIGHT,
        ACTION_1,
        LIB_NEXT,
        LIB_PREV,
        GAME_NEXT,
        GAME_PREV,
        MENU,
        QUIT,
};

using InteractionList = std::queue<Interaction>;
```

- `Interaction` enumeration list all the interactions handled by the Core of the Project.
- `InteractionList` is an `std::queue` of interactions.

Color

```
enum Color {
        BLUE,
        RED,
        GREEN,
        YELLOW,
        CYAN,
        MAGENTA,
        WHITE,
        BLACK,
        UNDEFINED,
        DFT_COLOR_RET_ERROR,
};
```

- `Color` enumeration is used to standardize the render of colors of non graphics libraries such as libcaca and ncurses.

Sprites & Items

```
	/* Sprites & Items */

	struct Sprite {
		int x;
		int y;
		int rotation;
		char substitute;
		std::string name;
		std::string path;
		Color color;
		Color background;
	};

	using SpriteList = std::vector<Sprite>;

	struct Item {
		std::string name;
		std::string spritesPath;
		SpriteList sprites;
		int currSpriteIdx;
		float x;
		float y;
	};

	using ItemList = std::vector<Item>;
}
```

- `Sprite` structure defines all the specifications of a texture/image such as:
    - `int x` and `int y` : position of the sprite relative to the Item one.
    - `int rotation` : between `0°` and `360°` set the orientation of the sprite.
    - `char substitute` : substitute of the image in non graphics libraries.
    - `std::string name` : Name for the Sprite **Not necessarily Unique** (can be used as an Sprite "type". e.g: "Pacgum", "Ghost", ...).
    - `std::string path` : Path to an image/texture of the Sprite.
    - `Color color` : color for the Sprite in non graphical libraries.
    - `Color background` : bacground color for the Sprite in non graphical libraries.
- `SpriteList` is an `std::vector` of Sprites.
- `Item` structure is the most important structure of this project. It gathers all the informations for an "Item" in order to be displayed:
    - `std::string name` : **Unique** name for an Item.
    - `std::string spritesPath` : Path to an image/texture of the Item. **Deprecated**
    - `SpriteList sprites` : All sprites for animation purpose.

    - `int currSpriteIdx` : Index to set the Sprite to use (automatically increment for animation).
    - `float x` and `float y` position of the item in the window (std::floor for non graphical libraries);
- `ItemList` is an `std::vector` of Items.

Time

```
/* Time */

using millisec = std::chrono::duration<double, std::milli>;
```

- `millisec` is an `std::chrono::duration<double, std::milli>` used for timers in games.

IGame interface

Interface to use in order to implement a new game:

```
/*
** EPITECH PROJECT, 2018
** cpp_arcade
** File description:
** IGame
*/

#ifndef IGAME_HPP_
	#define IGAME_HPP_

	#include <string>
	#include <vector>
	#include "IDisplay.hpp"

namespace arc {

class IGame {
public:
	struct Specs {
		int x;
		int y;
		uint pixelStep;
		uint fps;
	};

	virtual ~IGame() = default;
	virtual const ItemList &getItems() const noexcept = 0;
	virtual const Specs &getSpecs() const noexcept = 0;
	virtual bool processInteraction(Interaction &) noexcept = 0;
	virtual void envUpdate() noexcept = 0;
	virtual int getScore() noexcept = 0;
	virtual bool isOver() const noexcept = 0;
};
};

#endif /* !IGAME_HPP_ */
```

- The `Specs` structure is used to comunicate **game specifications**.
  - `int x` and `int y` : widht and height of the window.
  - `uint pixelStep` : Pixels on Graphicals libraries equals to one cell on a standart terminal. It should be set by your game.
  - `uint fps` : frame rate of the window.
- `const ItemList &getItems() const` : main fuction to use, it returns an `ItemList` composed of Items, it must be compose of all Items you need to display.
- `const Specs &getSpecs() const` : use to get the specs of a game.
- `bool processInteraction(Interaction &)` : Since the **Game Logic** is handeled by the Game, the `processInteraction` function is used to communicate the player inputs to the game one by one to procces them and move everything it needs accordingly. It return `true` on succes and `false` otherwise.
- `void envUpdate()` : called each loop to update all non playables items who need to procces an action.
- `int getScore()` : returns the Score of the player when called.
- `bool isOver() const` : return `false` by default and `true` when the game ends.

IDisplay interface

Interface to use in order to implement a new graphical interface:

```
/*
** EPITECH PROJECT, 2018
** cpp_arcade
** File description:
** IDisplay
*/

#ifndef IDISPLAY_HPP_
	#define IDISPLAY_HPP_
	#include "Arc.hpp"

namespace arc {

class IDisplay {
public:
	virtual ~IDisplay() = default;
	virtual void clear() = 0;
	virtual void refresh() = 0;
	virtual void putStr(const std::string &,
				int x = 0, int y = 0) = 0;
	virtual void putItem(const Item &) = 0;
	virtual void putItem(const Item &, int, int) = 0;
	virtual void putItem(const Item &,
		const std::vector<struct Position> &) = 0;
	virtual void setStep(uint) = 0;
	virtual InteractionList getInteractions() = 0;
};
}

#endif /* !IDISPLAY_HPP_ */
```

- `void clear()` : Fucntion to clear the window/terminal.
- `void refresh()` : Function to refresh the window/terminal.
- `void putStr(const std::string &, int x = 0, int y = 0)` : Function to display some text at a given position in the window. (Default 0;0).
- `void putItem(const Item &)` : polymorphic set of functions to use to display an Item in the window;
- `void setStep(uint)` : set the step.
- `InteractionList getInteractions()` : Function called by the Core to get all the keys pressed by The Player.