In this documentation you will find directions about how to add a new game or a new graphic library to the Core and make it compatible.

You will find on the first pages, an easy, simple documentation which will explain the Games & Libraries, then, you will get to find a documentation which will be more technical about the Interfaces and how they interact with each other.

EASY DOCUMENTATION:

- Add a new game

Your game will need to be compiled with GLIBC 2.31 or lower. It won't work with a higher version

To add a new game to the core for it to handle, there are a few requirements.
Firstly, you will need to name the Shared Object file like this:

(Note: GAME is to be replaced with the name of your game)
arcade_GAME.so

Then, you will have to add the game into the file "./Content/games.txt" like this:

(Note: GAME is to be replaced with the name of your game)
arcade_GAME

Once this is done, you will have to create a function called "entryPoint" into your game, which will return a shared_ptr of the instance of your class game (which must inherit from IGame())

Then the game must have a function called "update"
another function called "getDatas"
and a function "setInput"

setInput will be used to tell the game which button was pressed by the player, whereas update will be used to allow the game to run.

setInput will take an enum in IDisplayer named "keyboardKeys"

getDatas will return the map of the game in the form of a vector of string.

The first two lines of the map will have to contain the player's score and then health if there's any, if not, they must be empty with only a '\n'

Now that this is done, good job. You managed to add a game, all you have left to do is play it

- Add a new Library

Your library will need to be compiled with GLIBC 2.31 or lower. It won't work with a higher version

To add a new library to the core for it to handle, there are a few requirements.
Firstly, you will need to name the Shared Object file like this:

(Note: LIBRARY is to be replaced with the name of your game)
arcade_LIBRARY.so

Then, you will have to add the game into the file "./Content/libs.txt" like this:

(Note: LIBRARY is to be replaced with the name of your game)
arcade_LIBRARY

Once this is done, you will have to create a function called "entryPoint" into your game, which will return a shared_ptr of the instance of your class Displayer (which must inherit from IDisplayer())

Then the library must have a function called "drawGame"
another, named "drawMenu"
and a function "getInput"

getInput will return the key pressed by the player which is an enum present in IDisplayer

drawMenu will display the available games and libraries, which will be given as a vector of string (First games then Libraries)

drawGame will be used to draw the game, it takes a vector of string and will display it, as simple as that

Now, there are a few extra functions, we're very close.

You need to add in a "loadGameAsset" and a "loadMenuAsset"

loadMenuAsset is a void and takes a string, a vector of string and another vector of string as arguments.

loadGameAssets takes a string and a vector of strings.

Now that this is done, good job. You managed to add a game, all you have left to do is play it!

- Remove a game

You just need to remove it from the file ./Content/games.txt

- Remove a library

you just need to remove it from the file ./Content/libs.txt

TECHNICAL DOCUMENTATION:

Interfaces:

```
enum KeyboardKeys
{
    NONE,
    KEY_A,
    KEY_B,
    KEY_C,
    KEY_D,
    KEY_E,
    KEY_F,
    KEY_G,
    KEY_H,
    KEY_I,
    KEY_J,
    KEY_K,
    KEY_L,
    KEY_M,
    KEY_N,
    KEY_O,
    KEY_P,
    KEY_Q,
    KEY_R,
    KEY_S,
    KEY_T,
    KEY_U,
    KEY_V,
    KEY_W,
    KEY_X,
    KEY_Y,
    KEY_Z,
    ARROW_R,
    ARROW_U,
    ARROW_D,
    ARROW_L,
    NUMBER_1,
    NUMBER_2,
    NUMBER_3,
    NUMBER_4,
    NUMBER_5,
    NUMBER_6,
    NUMBER_7,
    NUMBER_8,
    NUMBER_9,
    NUMBER_0,
};
```

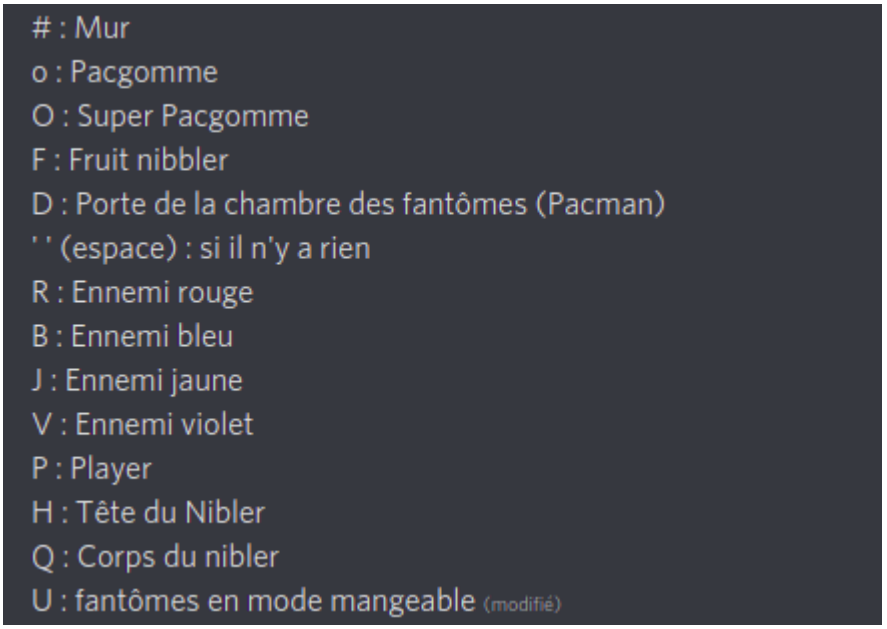
- Idisplayer:

The Interface handling the Displayer is in need of various things
There must be a DrawMenu, which will have to display the list of the Game's names we could load, and the graphic libraries.

Those are the only things that will be given by the Core as it loads the Shared Objects files and checks their names.

DrawGame is going to display the map of the game, in this map, there will be the necessary informations, such as:
Score
Health Points
The actual map

The graphic library only needs to display the map, the map consists of various characters with a set ruled characters:



loadMenuAssets is supposed to load every texture needed so that we save on the memory & execution time, preventing useless operations. It takes the current game selected as first argument, given by core, the list of all games we loaded, given by core, the list of all libraries given by core.

The sets of ruled characters are placed into the graphic Library

loadGameAssets is going to take the game's name in order to look for a subfolder in the folder "Assets", containing the game's assets in order to load the correct ones
it also needs the map to apply them on the correct location

- IGame:

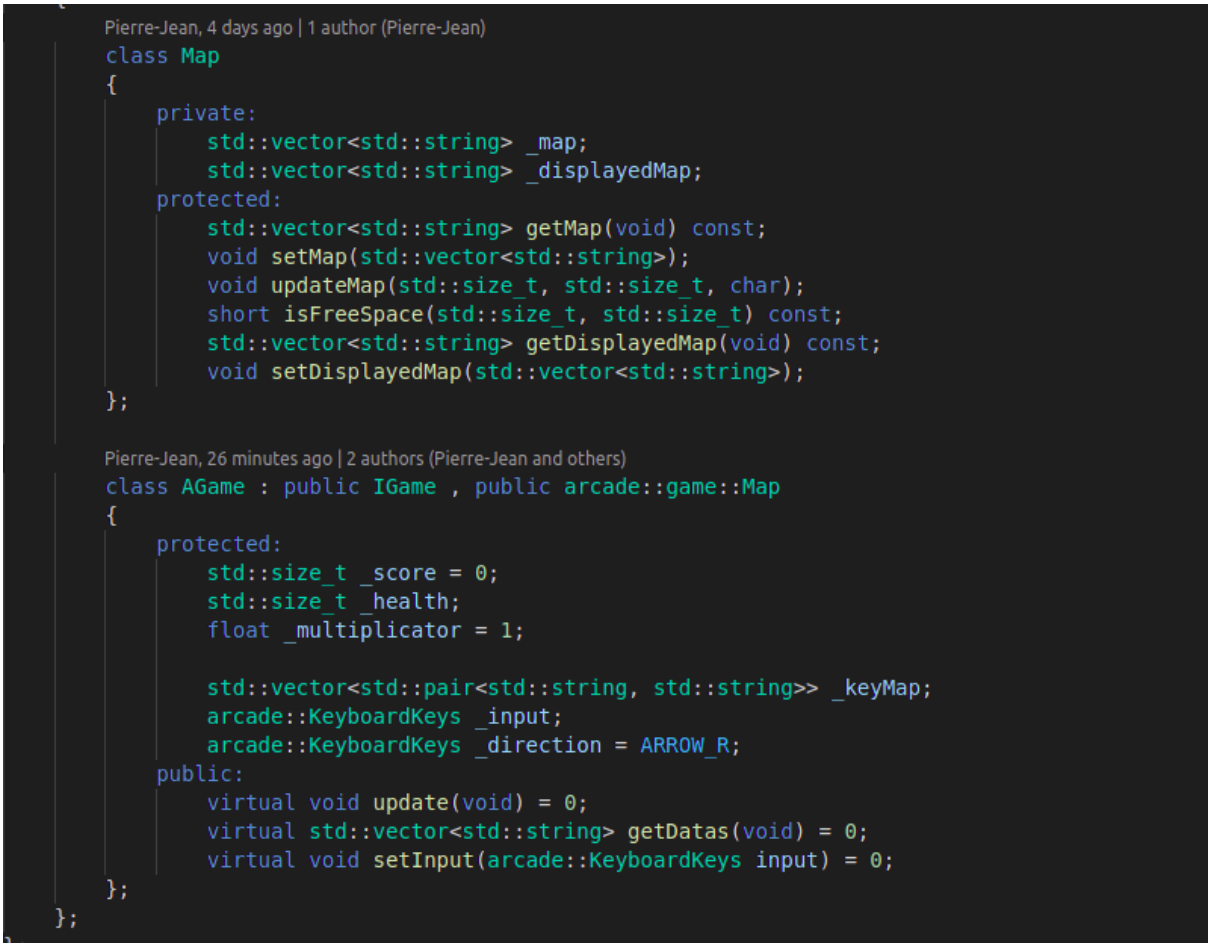
```
    virtual ~IGame() = default;

    virtual void update(void) = 0;

    virtual std::vector<std::string> getDatas(void) = 0;
    virtual void setInput(arcade::KeyboardKeys input) = 0;
};
```

the function update is the function that will make the game update itself, it updates the map and makes everything move ONCE.

the getDatas will return the map

the setInput will allow the game to know which input was pressed by the player

AGAME:



```
Pierre-Jean, 4 days ago | 1 author (Pierre-Jean)
class Map
{
    private:
        std::vector<std::string> _map;
        std::vector<std::string> _displayedMap;
    protected:
        std::vector<std::string> getMap(void) const;
        void setMap(std::vector<std::string>);
        void updateMap(std::size_t, std::size_t, char);
        short isFreeSpace(std::size_t, std::size_t) const;
        std::vector<std::string> getDisplayedMap(void) const;
        void setDisplayedMap(std::vector<std::string>);
};

Pierre-Jean, 26 minutes ago | 2 authors (Pierre-Jean and others)
class AGame : public IGame , public arcade::game::Map
{
    protected:
        std::size_t _score = 0;
        std::size_t _health;
        float _multiplicator = 1;

        std::vector<std::pair<std::string, std::string>> _keyMap;
        arcade::KeyboardKeys _input;
        arcade::KeyboardKeys _direction = ARROW_R;
    public:
        virtual void update(void) = 0;
        virtual std::vector<std::string> getDatas(void) = 0;
        virtual void setInput(arcade::KeyboardKeys input) = 0;
};
};
```

This is the Game's abstract, it sets it's own needed functions, being getMap setMap and others, which will be used by any game, they are as generic as possible

getMap / setMap, basic getter / setter for map
updateMap, updates a coordinate's char with another

isFreeSpace will basically tell if the asked coordinates are free space or not

getDisplayedMap basically returns the map entirely WITH the player's position & ennemies

setDisplayedMap allows to place the player's character & the ennemies on the map

_Score is here because every game got a score
_health is there, because every game offers to the player health (Be it one health point to symbolize that you'll lose instantly or multiple)

_keyMap is there to basically know which key was what

_input is here to know which input was pressed
_direction to know which direction the player is going

update, getDatas, setInput are coded in a very general manner.