B4 - Object-Oriented Programming

B-OOP-400

With :
Kévin PRUVOST,
Quentin FELIX,
Adrien HEROS

Arcade

A Retro Platform

2.0

+ HOW TO CREATE AND IMPLEMENT A NEW GRAPHICAL LIBRARY

You must use the interface *IDisplayModule* and *ArcadeContent* to code a new graphical library for our arcade machine and implement a dynamic instance creator named getDisplayModule, you can also use the SpriteData class to parse sprite informations.

The new display class prototype must at least have these functions :

```
class Sdl : virtual public ADisplayModule
{
public:
    Sdl(ArcadeContent & arcadeContent);
    ~Sdl();

    void updateEvent() override;
    void setTexture() override;
    void display() override;

private:
    std::vector<std::vector<char>> _oldMap;
};

extern "C"
{
    IDisplayModule * getDisplayModule(ArcadeContent & arcadeContent)
    {
        return new Sdl(arcadeContent);
    }
}
```

Then when the display class is created you will have to compile a dynamic library formated with lib_arcade_ at the beginning of the library's name, like these examples :

- lib_arcade_newDisplay.so
- lib_arcade_xXx_Display_xXx_.so

Then place the new display library in the lib/ folder of the arcade machine folder.

+ HOW TO CREATE AND IMPLEMENT A NEW GAME LIBRARY

You must use the interface *IGameModule* and *ArcadeContent* to code a new game for our arcade machine and implement a dynamic instance creator named getGameModule.

The new game class prototype must at least have these functions :

```
class PacMan : virtual public AGameModule
{
public:
    PacMan(ArcadeContent & arcadeContent);
    ~PacMan();

    void update() override;

private:
};

extern "C"
{
    IGameModule * getGameModule(ArcadeContent & arcadeContent)
    {
        return new PacMan(arcadeContent);
    }
}
```

Then when the game class is created you will have to compile a dynamic library formated with lib_arcade_ at the beginning of the library's name, like these examples :

- lib_arcade_newGame.so
- lib_arcade_xXx_Game_xXx_.so

Then place the new game library in the games/ folder of the arcade machine folder.

+ ARCADE CONTENT

```cpp
#define MAP_WIDTH  50
#define MAP_HEIGHT 30
#define SPRITE_WIDTH  (float)32
#define SPRITE_HEIGHT (float)32
#define WINDOW_X MAP_WIDTH  * SPRITE_WIDTH // 1600 = 50 * 32
#define WINDOW_Y MAP_HEIGHT * SPRITE_HEIGHT // 960 = 30 * 32

class ArcadeContent
{
public:
    ArcadeContent();
    ~ArcadeContent() = default;

public:
    std::vector<std::vector<char>> & map();
    ArcadeEvent & event();

private:
    std::vector<std::vector<char>> _map;
    ArcadeEvent _event;
};
```

ArcadeContent contains informations about the event system and the map to display.

The display library will have to draw what's written in the map and overwrite every happening events in the ArcadeContent.

The game library will have to write in the map what the display library should draw on the screen and use the event contained in ArcadeContent to make the game work.

+ ARCADE EVENT

```
class ArcadeEvent
{
public:
    enum KeyValue
    {
        NO_KVALUE = -1,
        LEFT, RIGHT, UP, DOWN,
        ESCAPE, ENTER,
        EXIT, MAINMENU,
        CHANGE_GAME, CHANGE_GRAPH
    };

public:
    ArcadeEvent(bool pressed = false, bool released = false,
                KeyValue keyValue = KeyValue::NO_KVALUE,
                int gameId = NO_KVALUE,
                int displayId = NO_KVALUE);
    ~ArcadeEvent();

    bool & isPressed()                      { return _pressed;   }
    const bool isPressed() const            { return _pressed;   }
    bool & isReleased()                     { return _released;  }
    const bool isReleased() const           { return _released;  }
    int & getKeyValue()                     { return _keyValue;  }
    const int getKeyValue() const           { return _keyValue;  }
    int & getGameId()                       { return _gameId;    }
    const int getGameId() const             { return _gameId;    }
    int & getDisplayId()                    { return _displayId; }
    const int getDisplayId() const          { return _displayId; }

private:
    bool _pressed = false;
    bool _released = false;
    int _keyValue = NO_KVALUE;
    int _gameId = NO_KVALUE;
    int _displayId = NO_KVALUE;
};
```

The ArcadeEvent class will mostly contain the key used and detected by the graphical library, the _gameId, and the _displayId.
The _gameId and the _displayId represent what game and display are currently used or what will be the next game or display if the _keyValue is set to CHANGE_GAME or CHANGE_GRAPH.

+ MAP

```
std::vector<std::vector<char>> & map();
```

```
std::vector<std::vector<char>> _map;
```

The map is contained in the ArcadeContent class, it is accessible by one of its method called map(), the purpose of this attribute is that it will contain what's displayed in the graphical library.

For our case, the map will contain a 50x30 character array of 2 dimensions (vector<vector<char>>), 50 of width and 30 of height.
Each character represents a sprite or 2 characters for textual graphical libraries like libcaca or ncurses. Each representation is stored in a resource file that we will see in the next page.

+ RESOURCE FILES

The resource file is a file linked to a game library.
It must be located at least in the games/ folder.
Taking games/ folder as a root point, you will have to specify the location of this resource file for a specific game.

Example :

```
MainMenu::MainMenu(ArcadeContent & arcadeContent)
    : AGameModule(arcadeContent, "resources/resourcesMainMenu")
```

If resourcesMainMenu is located in games/resources/resourcesMainMenu.

```
char(char) Sprite.png x(int) y(int) width(int) height(int) nb_anim(int) replaceChar(2char) COLOR_TEXT(int) COLOR_FOND(int)

# Sprite.png 0 0 800 800 1 () 1 1
# Sprite.png 0 0 800 800 1 () 1 1
# Sprite.png 0 0 800 800 1 () 1 1
# Sprite.png 0 0 800 800 1 () 1 1
# Sprite.png 0 0 800 800 1 () 1 1
```

The informations displayed in that file are formated like this :

- Identity Character,

For non-textual graphical libraries :
- Sprite picture file name,
- x coordinate of the wanted sprite if the Sprite picture file contains multiple sprites to use,
- y coordinate of the wanted sprite,
- Sprite width,
- Sprite height,
- Number of animations,

For textual graphical libraries :
- The 2 characters that will represent the sprite (2 because most of the time, terminals display characters with height twice as big as the width of the characters they display).
- Characters color.
- Charaters background color.

+ KEYBINDINGS

Keybinding	Action
Left Arrow / Q	LEFT
Right Arrow / D	RIGHT
Up Arrow / Z	UP
Down Arrow / S	DOWN
Escape	ESCAPE
Return / Space	ENTER
x	EXIT
M	MAINMENU
&	PREVIOUS DISPLAY MODE
é	NEXT DISPLAY MODE
"	PREVIOUS GAME
'	NEXT GAME

+ Classes Diagram