Documentation Arcade

Introducing the project

Arcade is a gaming platform, this program allows a player to choose the game he wants to play as well as the graphic library he wants to use.
We decided to implement 3 graphic libraries:

- NCurses
- SDL2
- SFML

We have also implemented 2 gaming libraries:

- Pacman
- Nibbler

Compilation of the program

The program is compiled with a Makefile:

- The *"make"* command at the root of the repo allows you to compile the entire project.
- The *"make core"* command at the root of the repo allows you to compile only the core of the project.
- The *"make games"* command at the root of the repo allows you to compile only the libraries of the implemented games.
- The *"make graphicals"* command at the root of the repo allows you to compile only the graphic libraries implemented.

Everything that is compiled by the Makefile is found in the folder. /lib at the root of the repo.

Launch the program

The program should be launched as follows:

« . /arcade ./lib/[NOM_DE_LA_LIBRAIRIE_GRAPHIQUE]. so »

This will launch the program with the original graphic library, the one of your choice.

How to play

1. On the menu

- The top and bottom arrow allow you to choose the game you want to play.
- You can enter your nickname using the different keys of the keyboard.
- The F1 button lets you load the previous graphics library.
- The F2 button lets you load the next graphic library.
- The Escape button lets you leave the program.

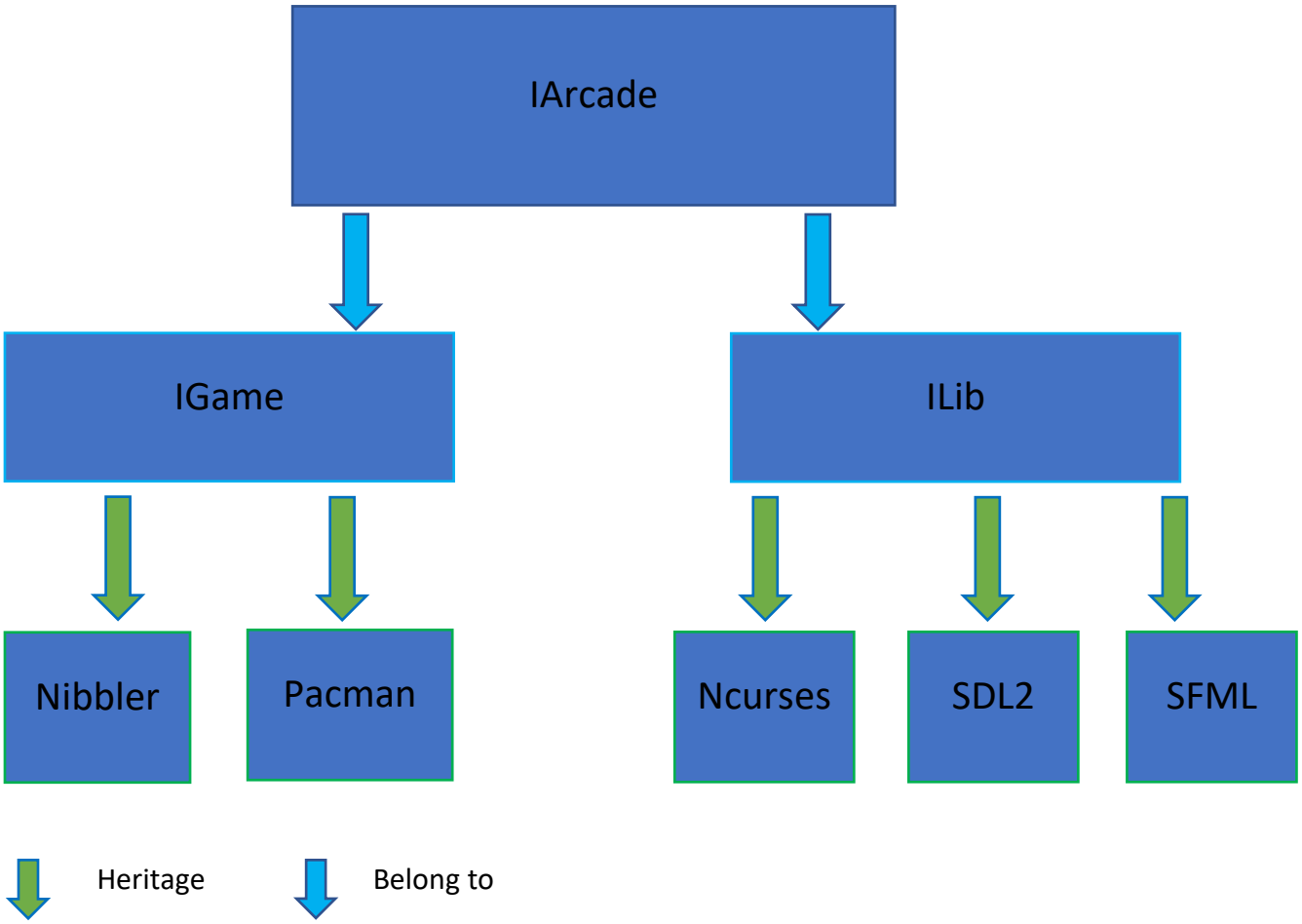
Once you are ready you can press the Entry button to launch your game.

2. In a game

- The arrows allow you to move around.
- The F1 button lets you load the previous graphics library.
- The F2 button lets you load the next graphic library.
- The F3 button lets you move on to the previous game.
- The F4 button lets you move on to the next game.
- The F5 button lets you start the game you are playing again.
- The F6 button lets you return to the menu.
- The Escape button lets you leave the program.

Development part

Decomposition of the project:



IArcade is an interface that contains a variable IGame and ILib. The Nibbler and Pacman interfaces inherit the IGame model interface to use the same functions for all games. It works the same way for graphic libraries and the ILib interface. Thanks to this similarity between interfaces, we can launch any game with any library.
The IArcade interface allows you to call in a loop:

- The library that picks up the input entered by the player.
- The game that reacts according to the input recovered before.
- The game that returns the list of all the items to draw on the window as well as their position.
- The library that draws one by one the items recovered before.

Decomposition of the model class of library game:

If you want to create a game, you must recode all the functions common to the games that are called by IArcade.

The red functions are the functions used by the IArcade interface.

- ## IGame :

```
Public :
   virtual ~IGames() = default;
   virtual void update(Key &key) = 0;
   virtual std::vector<Object> getObjectList() const = 0;
   virtual int move(Key &key) = 0;
   virtual int moveAlone() = 0;
   virtual int moveUp() = 0;
   virtual int moveDown() = 0;
   virtual int moveLeft() = 0;
   virtual int moveRight() = 0;
   virtual int getScore() const = 0;
   virtual int getLives() const = 0;
   virtual int getWave() const = 0;
   virtual void restart() = 0;
   virtual bool victory() = 0;
   virtual bool defeat() = 0;
   virtual int looseLife() = 0;
 protected :
   clock_t _clock;
   std::vector<Object> _objectList;
   std::vector<std::string> _map;
   int _input;
   int _score;
   int _direction;
   Vect2i _pos;
   bool _infinLoop;
   int _wave;
   int _lives;
   float _speed;
```

- ➔ **update(Key -key)** takes a Key in setting that contains the input entered by the player and makes the game react with a loop turn according to this input.
- ➔ **getObjectList()** this feature returns the Vector of *Object* created by the game that contains all that there is to draw on the window with their positions.
- ➔ **getScore** returns the score that the player is making during his game.
- ➔ **getLives()** returns the number that the player currently has during his game.
- ➔ **getWave returns** the number of the wave in which the player is located.
- ➔ **restart()** restart the game in its original state.

In order to illustrate the games by any graphic library, the games create a vector composed of the structure *Object_t.*

```
typedef struct Object_t
{
  int objectID;
  std::string texturePath;
  Vect2f fPos;
  Vect2i iPos;
  Rectangle rect;
  const char *character;
  int colorID;
  Vect2f fOffSet;
  char mapChar;
} Object;

typedef struct Rectangle_t
{
  int top;
  int left;
  int width;
  int height;
} Rectangle;

typedef struct Vect2i_t
{
  int x;
  int y;
} Vect2i;

typedef struct Vect2f_t
{
  float x;
  float y;
} Vect2f;
```

This vector contains everything a graphic library needs to illustrate the player's actions on the game.

If you want to create a game compatible with our project, it is necessary that your game create a vector of*Object_t* and that it be returned to the function: std::vector<Object> getObjectList() const.
You must have the common functions of our games like: *update, getScore, getLives...*
Finally, your game will have to be compiled in dynamic library (. so) and add in the folder. /lib at the root.

Decomposition of the graphic library model interface

If you want to create a graphic library, you need to recode all these functions.

- ILib :

```
public:
  virtual ~ILib() = default;
  virtual void init() = 0;
  virtual void displayObject(Object &object) = 0;
  virtual void displayBackground() = 0;
  virtual void displayWindow() = 0;
  virtual void destroyWindow() = 0;
  virtual void putText(Text &text) = 0;
  virtual void createObjects(int &gameID) = 0;
  virtual void getInput(Key &input) = 0;
  virtual std::string getLibName() = 0;
  virtual void clearWindow() = 0;
  virtual void destroyObjects() = 0;
```

The redfonctions, are the functions used by the IArcadeinterface.

- ➔ **init( )** initiates the graphic library.
- ➔ **displayObject(Object-object)** retrieves an *Object* (see above) and draws it on the window.
- ➔ **displayBackground()** draws the background every time it is called.
- ➔ **displayWindow()** makes the bookseller's window appear.
- ➔ **destroyWindow** destroys the window as well as the elements initiated by the **init function.**
- ➔ **putText(Text andtext)** retrieves a *Text* (see below) and draws it on the window.
- ➔ **createObjectf (gameID)** recovers an ID in setting and depending on this will load a texture.
- ➔ **getInput(Key-input)** retrieves a *Key* (see below) in setting and fills it according to the key entered by the player.
- ➔ **getLibName()** returns the name of the Lib.
- ➔ **clearWindow()** va clear la Window.
- ➔ **destroyObject()** will destroy the texture that was created by *createObject(int -gameID).*

```
typedef struct Text_t
{
  std::string fontPath;
  std::string text;
  Vect2i intPos;
  Vect2f floatPos;
  int textID;
  int color;
  int size;
} Text;
```

```
typedef struct Key_t {
  KeyState state;
  KeyCode code;
} Key;

typedef enum KeyState_t {
  StateNone,
  KeyPressed,
  KeyReleased
} KeyState;
```

Keycode is an enum that includes all the keys that are usable by the player.

In order to create a new graphic library that can be used by our project, it is necessary that your library have the functions common to all libraries.
Your graphic library may of course use other features.
Warning: these functions should not change the link between IArcade and your ILib.
In addition, it must use the information available in the vector of *Object_t*. Finally, your graphic library will have to be compiled in .so and add in the folder. /lib at the root.

Your Library must of course be equipped with a Makefile, which must be called by the Makefile at the root.

This project was carried out by:

- Titouan Loeb : titouan.loeb@epitech.eu
- Ilan Benarroche : ilan.benarroche@epitech.eu
- Julien Fumey : julien.fumey@epitech.eu