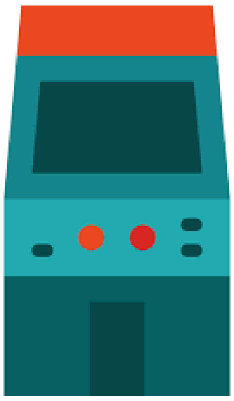
B4 - C++ Programming

B-CPP-400

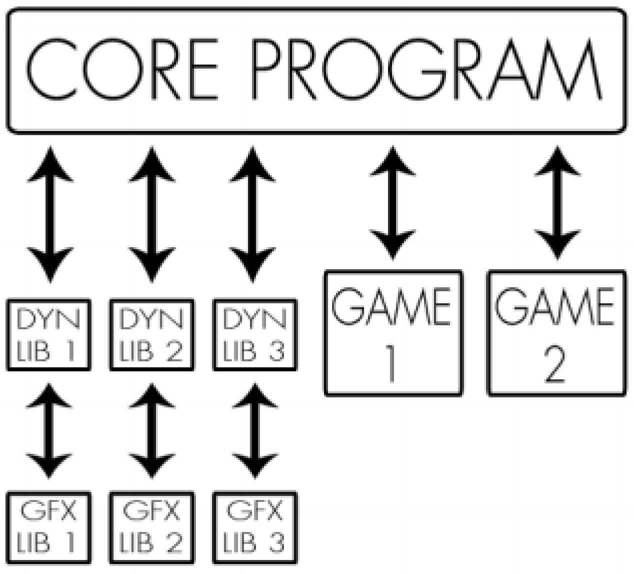
Arcade Documentation

A Retro Platform



THE 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 platform at run-time, the graphic libraries and the 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.



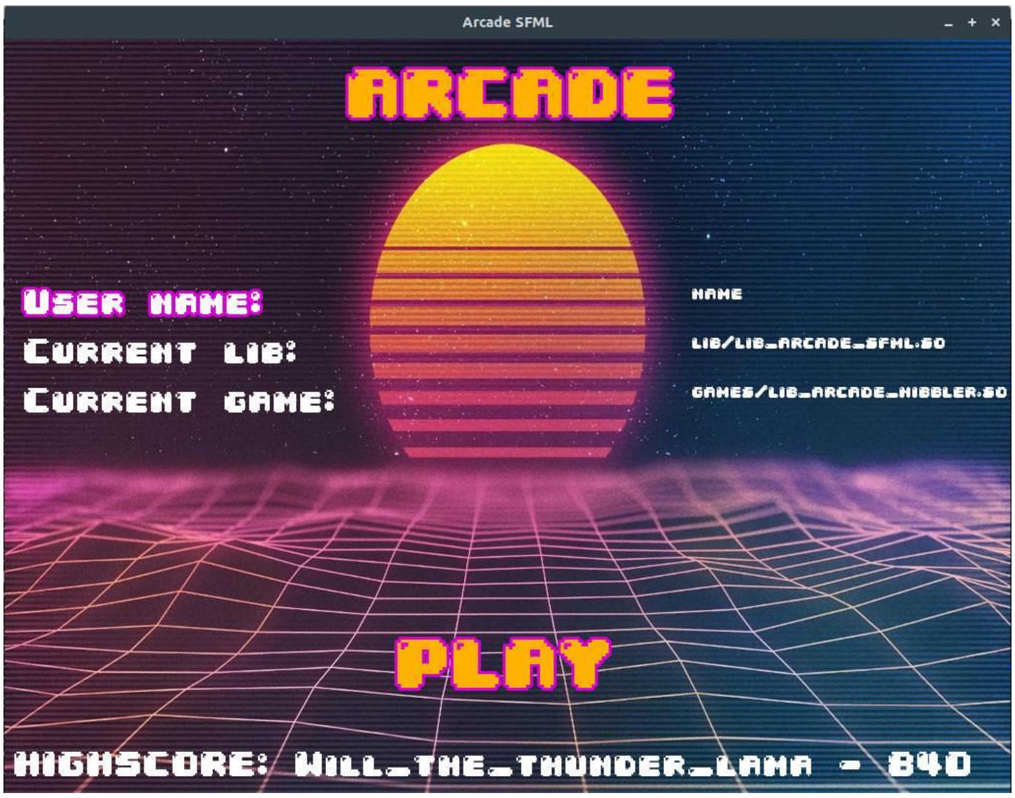
HOW TO USE

To launch the arcade, you must compile the program with **make**. To launch the program, you can do like in this example:

```
~ λ ./arcade ./lib/lib_arcade_libname.so
```

The program takes as parameter the graphics library to use initially, located in "./lib".

When the program is launched, you start in the main menu (as the provided example below), where you can select the game to play, the graphics library you want to use and the player name. It shows the highest score of the selected game (if none, it just displays 0).



Arcade SFML

ARCADE

USER NAME: NAME

CURRENT LIB: LIB/LIB_ARCADE_SFML.SO

CURRENT GAME: GAMES/LIB_ARCADE_NIBBLER.SO

PLAY

HIGHSCORE: WILL_THE_THUNDER_LAMA - 840

To navigate the menu, you can use the **arrow keys on the keyboard**. When *USER NAME* is selected, you can modify the name using the **keyboard** (**backspace** to delete characters). When *CURRENT LIB* is selected, you can use the **right** and **left** arrow keys to change the current graphics lib. Finally, when *CURRENT GAME* is selected, you can use the **same keys** as last step to change the current game (the highest score should update too).

When everything is ready, you can select the *PLAY* button, then press **enter/return** key to launch the selected game with the selected graphics library.

At any moment, you can press the escape key to quit the program.

Within each game, you can use some keys to change parameters. You can find what they do below:

- *A*: previous game
- *Z*: next game
- *E*: previous graphics library
- *R*: next graphics library
- *T*: restart the current game
- *Y*: get back to the menu (quits the game)

HOW TO ADD NEW CONTENT

As stated in the introduction, you can **add your own** graphics library and games to the program. To achieve this, you must follow two interfaces (one for the graphics, one for the games) to implement these "**plugins**".

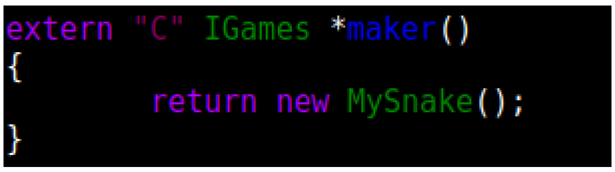
When this is done, you can add the sources of your plugin into the **Makefile** to compile it to a dynamic library. In the following sections, there will be a guide to create a **new graphics library** and a **new game** for the *arcade*.

+ ADD A NEW GAME

To add a new game to the arcade, you have first to implement the IGames.hpp interface given in "./inc". It contains the following functions to create:

- *void init_game()* :
  inits/reset every variable that will be used in the game (for example: the map, player, scores, etc...)
- *std::string get_asset_path()* :
  returns the path for the assets of the game (placed in "./assets", example: "./assets/my_game")
- *bool loop()* :
  returns a bool, true if the game is running, false if the game isn't (game lost/won)
- std::vector<std::string> get map() :
  returns the map, formatted as a vector of strings (each iteration of the vector contains a line of the map)
- *int get_score()* :
  returns the score
- *void manage_key(Input)* :
  gets the input of the user as an Input enum (definition given in IGraphical.hpp)
- *void quit()* :
  you can free everything here if needed

In your code, you must create a ***"maker"***, so the arcade can instantiate your game class. Here an example for a snake class:



```
extern "C" IGames *maker()
{
        return new MySnake();
}
```

+ ADD A NEW GRAPHICS LIBRARY

To add a new graphics library to the arcade, you have first to implement the IGraphical.hpp interface given in "./inc". It contains the following functions to create:

- *void init_window()*
  everything needed to create and open a window
- *int is_open()*
  returns a bool telling if the window is open

- *void close_window()*
  closes the window
- *Input get_key_pressed()*
  returns an Input enum corresponding to the pressed key (arrow keys + esc + enter/return), returns UNDEFINED if none of these
- *int check_event()*
  returns a bool telling if an event happened (pollEvent for example)
- *int check_close_window()*
  returns if the window received a close signal (event closed)
- *void prepare_close_window()*
  sets the window to "not open"

- *void display_wd()*
  displays the current window
- *void clear_wd()*
  clears the current window

- *void init_menu()*
  here you can create/assign the sprites/fonts that'll be used in the menu (to avoid constantly creating/assigning the sprites every frame)

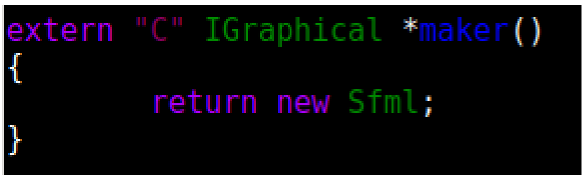
For the following functions, you must draw the sprites/text you need, but not to display nor clear the window:

- *void draw_title_arcade()*
  draws in the window the title ("ARCADE" for example)
- *void draw_user(const char*, Menu_selec)*
  draws the user name (1st argument). if 2nd argument is equal to USER, then this option is selected in the menu (so you can highlight the option)
- *void draw_current_lib(const char*, Menu_selec)*
  draws the name of the current lib (1st argument). if 2nd argument is equal to LIB, then this option is selected in the menu (so you can highlight the option)
- *void draw_current_game(const char*, Menu_selec)*
  draws the name of the current game (1st argument). if 2nd argument is equal to GAME, then this option is selected in the menu (so you can highlight the option)
- *void draw_play(Menu_selec)*
  draws the play button (text), if 1st argument is equal to PLAY, then this option is selected in the menu (so you can highlight the option)
- *void draw_highscore(std::string, int)*
  draws the highest score. 1st argument is the player name, 2nd argument is the score

- *std::string get_pseudo_user(std::string)*
  1st argument is the current pseudo: you must return the new pseudo (this function gets the key pressed by the user on the keyboard and modifies the string, you must get backspace to delete a character)

- *void init_graphical(std::string)*
  gets all the sprites needed for the game, located in "./assets/game_name" (this is the first argument). You need to add to this string the name of your graphics library (example: "./assets/pong/Unreal Engine 4"), the graphics library specific sprites will be here
- *int get_index(char)*
  the 1st argument is a char of the map given (for example: map[7][2]). The function must return the index of the char in the Env enum (example: argument = ' ', then it's VOID which is the index 0, return 0. other example: argument = 'e', ENEMY is index 4, return 4). The function returns -1 if the char is unknown
- *void draw_game(std::vector<std::string>)*
  draws the map of the game, using the sprites (if needed). The map is given as the vector of strings (each iteration of the vector is a line of the map). Access the map with map[y][x]
- *void draw_score(int)*
  draws the score, given as parameter
- *Event get_event()*
  returns an Event enum, corresponding for one of the following keys: A, Z, E, R, T, Y, Escape. Returns NO_EVENT if none of these. Refer to the *HOW TO USE* section to know which key corresponds to which action

In your code, you must create a ***"maker"***, so the arcade can instantiate your game class. Here an example for a snake class:



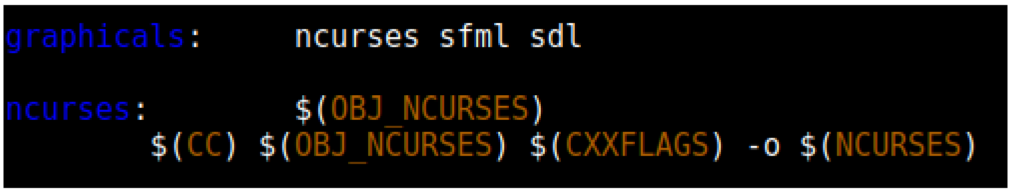
```
extern "C" IGraphical *maker()
{
        return new Sfml;
}
```

+ RECOMANDATIONS

In this section, you will find some tips and warning about adding plugins to the arcade.

- assets must be in a specific folder, with specific names.
  - ./assets/graphics_library contains the background for the menu. The file must be a courant extension for images (jpeg, png, jpg, etc…), nothing too fancy, and must be sized 1024x770
  - ./assets/game/graphics_library contains the sprites for the game and graphics library. These must be sorted in a way that it respects the Env enum order (first image must be the void, second the wall, etc…). They must be sized 128x128 (usually rescaled to 0.1 at display time), and same as above for the extension
- fonts are in ./fonts and must be ttf files. They must be named after a graphics library (for example: ./assets/Frostbite3.ttf)
- graphics libraries are in ./lib
- games are in ./games
- If you don't respect the rules above, the program might be not working properly or doing some weird things (maybe even crash every computer within 100 meters or so)

- to add your plugin to the Makefile, you must:
    - create a name variable (./lib/lib_arcade_[name].so or ./games/lib_arcade_[name].so), the folder and the “lib_arcade_[…].so” part is important, don’t touch it, or consequences
    - add the libraries that needs to be linked in CXXFLAGS
    - add your sources in a “SRC_LIBNAME” variable
    - do some *magik* trick in a “OBJ_LIBNAME” variable (like *$(SRC_LIBNAME:.cpp=.o)*)
    - create a rule named after your lib, that will do the OBJ part, then compile it to a dynamic library (example below)



```
graphicals:     ncurses sfml sdl

ncurses:        $(OBJ_NCURSES)
        $(CC) $(OBJ_NCURSES) $(CXXFLAGS) -o $(NCURSES)
```

    - add your rule to the corresponding rule (graphicals or games)
    - add to the clean rule OBJ_LIBNAME, to the fclean rule LIB_NAME
    - profit
- You can use *make graphicals* to compile only the graphics libraries, *make games* for the games or *make core* for the core

- Don’t forget to make your lib’s classes inherit from the respective interfaces !

Feel free to send a mail to samuel.rousseaux@epitech.eu, or william.tessari@epitech.eu, or tahery_b@epitech.eu for any information concerning this documentation or the project in general ! *(otherwise, git gud)*