Arcade Documentation

Sommaire

Introduction

The arcade is a 2nd year project of Epitech. The aim is to create an arcade born with 2 games and 3 graphical libraries.

The arcade must be able to change betwwen games and graphical libraries at runtime while saving player's position for example.

The project is given with a Nibbler and a Solarfox for games, and a graphical implementation of NCurse, SFML and OpenGL libraries.

Compilation

The project is given with a Makefile containing multiple rules to compile different parts of the project like the arcade binary, or games libraries.

Here is an array containing every compilation possibilities :

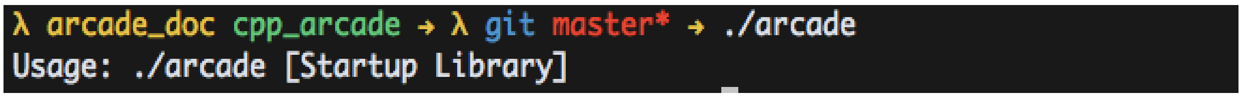
Rules	Action
all	Compile arcade and games
clean	Delete .o files from the arcade and games
fclean	Delete .o files and binaries from the arcade and games
re	Call fclean then all
global	Compile arcade, games and graphical parts
global_clean	Delete .o files
global_fclean	Delete .o files and created binaries
global_re	Call global_fclean then global
games	Compile games
games_clean	Delete .o files from games
games_fclean	Delete .o files and game binaries
games_re	Call games_fclean then games
snake	Compile Snake / Nibbler
snake_clean	Delete .o files from Snake
snake_fclean	Delete .o files and Snake binary
snake_re	Call snake_fclean then Snake
solar	Compile SolarFox
solar_clean	Delete .o files from SolarFox
solar_fclean	Delete .o and SolarFox binary
solar_re	Call solar_fclean then solar
graph	Compile graphical libraries
graph_clean	Delete .o files from graphical libraries
graph_fclean	Delete .o files and graphical binaries
graph_re	Call graph_fclean then graph
opengl	Compile OpenGL
opengl_clean	Delete .o files de la OpenGL
opengl_fclean	Delete .o files and OpenGL binary
opengl_re	Call opengl_fclean then opengl
sfml	Compile SFML
sfml_clean	Delete .o files from SFML
sfml_fclean	Delete .o files and SFML binray
sfml_re	Call sfml_fclean then sfml
ncurses	Compile NCurse
ncurses_clean	Delete .o files from NCurse
ncurses_fclean	Delete .o files and NCurse binary
ncurses_re	Call ncurse_fclean then ncurse

Finally, the project needs dependencies to be compiled with success:

- Lib SFML
- Lib SDL
- Lib NCurse
- Lib OpenGL
- Lib GLU
- Lib GLFW
- Lib GLEW
- Lib GLFT
- Lib DevIL
- Lib Assimp

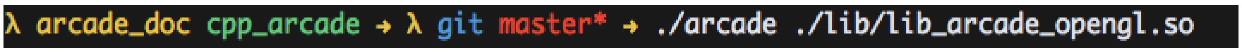
Basic usage

In a terminal, execute the following command after using the Makefile for the compilation:



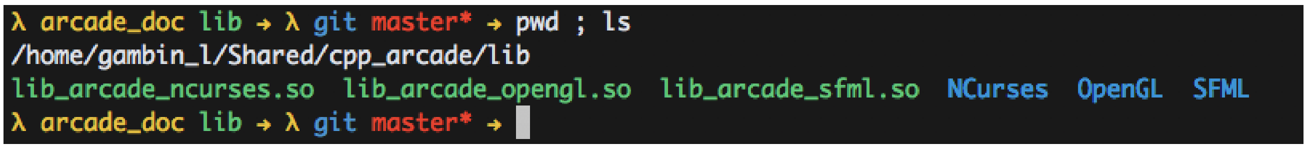
```
λ arcade_doc cpp_arcade → λ git master* → ./arcade
Usage: ./arcade [Startup Library]
```

The binary needs the default graphical library :



```
λ arcade_doc cpp_arcade → λ git master* → ./arcade ./lib/lib_arcade_opengl.so
```

Dynamic libraries must be in the "./lib/ " to be used in the arcade. Graphical libraries must be in the "./games/".



```
λ arcade_doc lib → λ git master* → pwd ; ls
/home/gambin_l/Shared/cpp_arcade/lib
lib_arcade_ncurses.so  lib_arcade_opengl.so  lib_arcade_sfml.so  NCurses  OpenGL  SFML
λ arcade_doc lib → λ git master* →
```

Program architecture

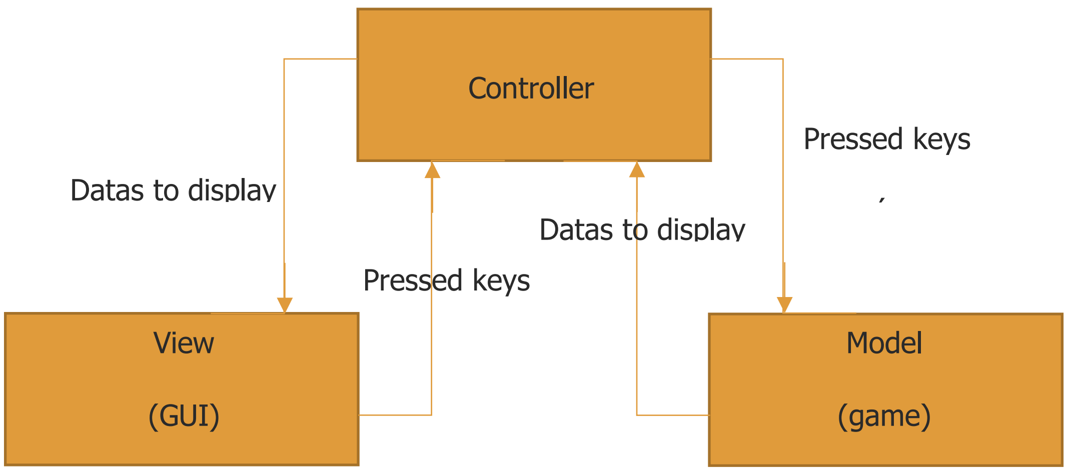
Arcade, when it starts, loads games and graphical libraries.

Each external entity must be a dynamic shared library ".so". You will be able to add your games and graphical libraries if you implement our interfaces.

The program is composed of 3 parts :

- The arcade, this binary is the core of the proram, this is the "controller". Informations will pass through it.
- Games or menu gives to the « controller » elements that must be displayed by the other graphical part.
- Graphical libraries receive from the "controller" elements to display on the screen and return pressed keys.

  Our program follows a MVC pattern.



Games must follow our interfaces with all member functions. Of course, you game must inherit of our interface. Same for graphical libraries

Data passing between model and view follows to a precise interface. From this interface, there is 3 parts:

- Visual datas are data that will be displayed on the screen, they inherit from "AVisual".
- Then there is datas about 3D scenes, they inherit from "AScene".
- Finally, a class is here to create audio objects.

Implementation norm

The aim of this part is to explain how to create a game or a GUI that will be compatible with our arcade binay.

First, to create a game, you must create the principal class of your game, this one will inherit from out interface "IGame" and you must implement virtual pure methods.

Our controller transmit datas thanks to a « std::vector<AData*> ».

This vector will come from game, pass through the "controller" and go to the graphical part.

Your extension must be compiled as a dynamic shared library and be compatible with the system that compiled the arcade.

Create a game

Your game must stock this vector to push back into datas.

Your game's class must have 16 member functions that the "controller" will use:

- **void InitGame();**
  - This method initializes your game once the « controller » takes an instance to your game after setting a ScoreManager.
- **std::string const &getGameName() const;**
  - This method returns the game's name.
- **int getFrameRatePerSecond() const;**
  - This method returns the maximum FPS.
- **std::vector <AData *> const &getData() const;**
  - This method returns the data vector that will be interpreted by the graphical part.
- **std::vector <std::string> const & getSprite() const;**
  - This method is here to optimize the sprite loading.

- **std::vector <std::string> const & getMusic() const;**
    - This method is here to optimize the musics loading.
- **std::vector <std::string> const & getSModel3D() const;**
    - This method is here to optimize the 3D models loading.
- **void setScoreManager(ScoreManager *scoreManager);**
    - The « controlller » will give to your game a ScoreManage before initialize this one.
- **void updateNewScore() const;**
    - This method must call pushNewScore(int) of the ScoreManager.
- **void play();**
    - This method calculate a frame of your game.
- **void getMap();**
    - This method write on the standard output your map for unit tests.
- **void whereIAm();**
    - This method write on the standard output your player's position for unit tests.
- **void goUp();**
    - This method is called when the graphical part detects the key to move the player up.
- **void goDown();**
    - This method is called when the graphical part detects the key to move the player down.
- **void goLeft();**
    - This method is called when the graphical part detects the key to move the player left.
- **void goRight();**
    - This method is called when the graphical part detects the key to move the player right.
- **void goForward();**
    - This method is called when the graphical part detects the key to boost the player.
- **void shoot();**
    - This method is called when the graphical part detects the key to shoot.

Create a GUI

As the game, a GUI must use member function of IGraph interface to work.

- **void InitLib();**
  - This method will initialize your graphical library.
- **std::string const & getLibName() const;**
  - This method returns the library name.
- **void giveData(std::vector <AData *> const &data);**
  - This method receives the data vector from the controller.
- **void giveSprite(std::vector <std::string> const &spriteList);**
  - This method receives a vector of sprites path.
- **void giveMusic(std::vector <std::string> const &spriteList);**
  - This method receives a vector of music path.
- **void giveModel3D(std::vector <std::string> const &spriteList);**
  - This method receives a vector of 3D Model path.
- **void setBridge(IArcadeBridge * bridge);**
  - This method is used by the controller to give an instance to it. You will be able to use methods of the controller from your library.
- **void handleData(AData const & data);**
  - This method must call the good display member function for the data in parameter.
- **void handleSphere(AData const & data);**
  - This method must display the data sphere given in parameter.
- **void handleCube(AData const & data);**
  - This method must display the data cube given in parameter.
- **void handleCamera(AData const & data);**
  - This method must display the data camera given in parameter.
- **void handleLight(AData const & data);**
  - This method must display the data light given in parameter.
- **void handleMusic(AData const & data);**
  - This method must display the data music given in parameter.
- **void handleText(AData const & data);**
  - This method must display the data text given in parameter.
- **void toggleRunning() const;**
  - This method is not implemented
- **void prevGraph() const;**
  - This method notifies the controller that the previous graph must be use.
- **void nextGraph() const;**

    - This method notifies the controller that the next graph must be use.
- **void prevGame() const;**
    - This method notifies the controller that the previous game must be use.
- **void nextGame() const;**
    - This method notifies the controller that the next graph must be use.
- **void goUp() const;**
    - This method notifies the controller that the Up key is pressed.
- **void goDown() const;**
    - This method notifies the controller that the Down key is pressed.
- **void goLeft() const;**
    - This method notifies the controller that the Left key is pressed.
- **void goRight() const;**
    - This method notifies the controller that the Right key is pressed.
- **void goForward() const;**
    - This method notifies the controller that the Enter key is pressed.
- **void shoot() const;**
    - This method notifies the controller that the Space key is pressed.
- **void pressEchap() const;**
    - This method notifies the controller that the Escap key is pressed.
- **void pressEight() const;**
    - This method notifies the controller that the 8 key is pressed.
- **void pressNine() const;**
    - This method notifies the controller that the 9 key is pressed.