Arcade (Pauline Fauré, Alexandre Flion, Maximilien Nadji)

Summary

About

On this document, you will find the explanations about how our program and our interfaces work. (Our project works only on Linux)

Starting

In order to start the program from the terminal, create a folder called build at the root of the directory.

```
[nmaximilien@desktop-na5no9s B-OOP-400-TLS-4-1-arcade-pauline.faure]$ mkdir build
```

Then from this folder, start the Makefile generation.

```
[nmaximilien@desktop-na5no9s B-OOP-400-TLS-4-1-arcade-pauline.faure]$ mkdir build
[nmaximilien@desktop-na5no9s B-OOP-400-TLS-4-1-arcade-pauline.faure]$ cd build/
[nmaximilien@desktop-na5no9s build]$ cmake ..
```

Next, compile the program and get back to the root and start the program as follows.

```
[nmaximilien@desktop-na5no9s B-OOP-400-TLS-4-1-arcade-pauline.faure]$ mkdir build
[nmaximilien@desktop-na5no9s B-OOP-400-TLS-4-1-arcade-pauline.faure]$ cd build/
[nmaximilien@desktop-na5no9s build]$ cmake ..
-- The CXX compiler identification is GNU 9.3.1
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ - works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/nmaximilien/Epitech/Tek2/B-OOP-400-TLS-4-1-arcade-pauline.faure/build
[nmaximilien@desktop-na5no9s build]$ make
Scanning dependencies of target arcade_sdl2
[  8%] Building CXX object CMakeFiles/arcade_sdl2.dir/src/SDL2.cpp.o
[ 16%] Linking CXX shared library arcade_sdl2.so
[ 16%] Built target arcade_sdl2
Scanning dependencies of target arcade_nibbler
[ 25%] Building CXX object CMakeFiles/arcade_nibbler.dir/src/Nibbler.cpp.o
[ 33%] Linking CXX shared library arcade_nibbler.so
[ 33%] Built target arcade_nibbler
Scanning dependencies of target arcade
[ 41%] Building CXX object CMakeFiles/arcade.dir/src/main.cpp.o
[ 50%] Building CXX object CMakeFiles/arcade.dir/src/Core.cpp.o
[ 58%] Building CXX object CMakeFiles/arcade.dir/src/Scene.cpp.o
[ 66%] Building CXX object CMakeFiles/arcade.dir/src/LibraryHandler.cpp.o
[ 75%] Building CXX object CMakeFiles/arcade.dir/src/Entity.cpp.o
[ 83%] Building CXX object CMakeFiles/arcade.dir/src/AGame.cpp.o
[ 91%] Linking CXX executable arcade
[ 91%] Built target arcade
Scanning dependencies of target move
[100%] Built target move
[nmaximilien@desktop-na5no9s build]$ cd ../
[nmaximilien@desktop-na5no9s B-OOP-400-TLS-4-1-arcade-pauline.faure]$ ./arcade ./lib/arcade_sdl2.so
```

API

Before explaining interfaces, let's talk about the API.

This API consists of having functions which allow you to:

- Create the library given in parameter at the launch of the program
- Delete the library when the program closes
- Recover the data of the library.

The API is in the api.h file inside the src folder.

Library management

The management of the libraries is carried out from the LibraryHandler class located at the src folder. This class allows to check if at the start of the program, the user gives an existing graphic library.

Interfaces

Now, let's talk about interfaces.

There are a total of seven interfaces. Here they are with their descriptions:

- ILibrary: represents game libraries and graphic libraries (located in the src folder)
- IGraphic: represents graphics libraries (located in the src folder)
- IGame: represents games libraries (located in the src folder)
- IScene: contains the scene in which the games and graphics libraries interact. (located in the src folder)
- IEntity: contains elements which composing the scene (located in the src folder)
- IEvent: represents game events and graphic events (located in the event folder)
- IComponent: contains properties / attributes of entities (located in the component folder)

Classes

About classes, here's the list:

MainMenu

The MainMenu class represents the MainMenu of the Arcade. It allows to run the game library and the graphic library simultaneously, to intercept an event, translate them (if it is a keyboard event or if it is a mouse event) and send this event to the game.

Here are its methods:

- run: start the main menu.
- onMouseEvent: process the mouse events.
- onKeyboardEvent: process the keyboard events.

AGame

The AGame class represents in an abstract way games libraries. It includes two event stacks: the first one for the mouse event and the second one for the keyboard event. It implements the IGame interface.

Here's its methods:

- onMouseEvent: process the mouse events.
- onKeyboardEvent: process the keyboard events.

NCurses

The NCurses class represents the ncurses library. It implements the IGraphic interface.
Here are its methods:

- init: initialize the ncurses and scene entities.
- update: update the ncurses according to the events and the scene entities.
- end: destroy scene entities and the ncurses.
- quitRequested: quit the program.
- handleEvents: manage events.
- displaySprites: display sprites.

Nibbler

The Nibbler class represents the Nibbler game. It inherits the AGame class.

Here are its methods:

- init: initialize the game and scene entities.
- update: update the game according to the events and the scene entities.
- end: destroy scene entities and the game.
- getScore: return the score.
- getHighScore: return the high score.
- getMap: return the map.
- getPos: return the position of the snake.
- updateSnakePos: update the position of the snake.
- parseMap: analyse the map.
- manageScore: update the score.
- stockHighScore: stock the high score.
- displayData: display data in game.
- displayMap: display the map.

Entity

The Entity class represents elements which compose the scene. It implements the IEntity interface.
Here are its methods:

- addComponent: allows to add a component to an entity.
- removeComponent: allows to remove a component to an entity.
- forEach: goes through all components and execute a function.
- getName: return the name of the entity.
- getPos: return the position of the entity.
- setPos: initialize the position of the entity.

Scene

The Scene class represents game scene. It implements the IScene interface.

Here are its methods:

- exit: exit the scene.
- pushEvent: send an event to the Core.
- newEntity: create a new entity. The entity can be created with a name.
- getEntity: return a collection of entities.
- removeEntity: remove an entity by its name or by its type.
- addScore: add the score.
- forEach: goes through all entities and execute a function. It can go through all entities according to a name.
- setWindowSize: initialize window coordinates.
- getWindowSize: return the size of the window.
- forEachEvents: goes through keyboardEvent and mouseEvent and execute a function.
- clearEvents: erase the content of the event stack.
- running: allows to know if a scene is running.

Credits

Pauline Fauré: https://github.com/paulinefaure
Alexandre Flion: https://github.com/huntears

Maximilien Nadji: https://github.com/max-ndj