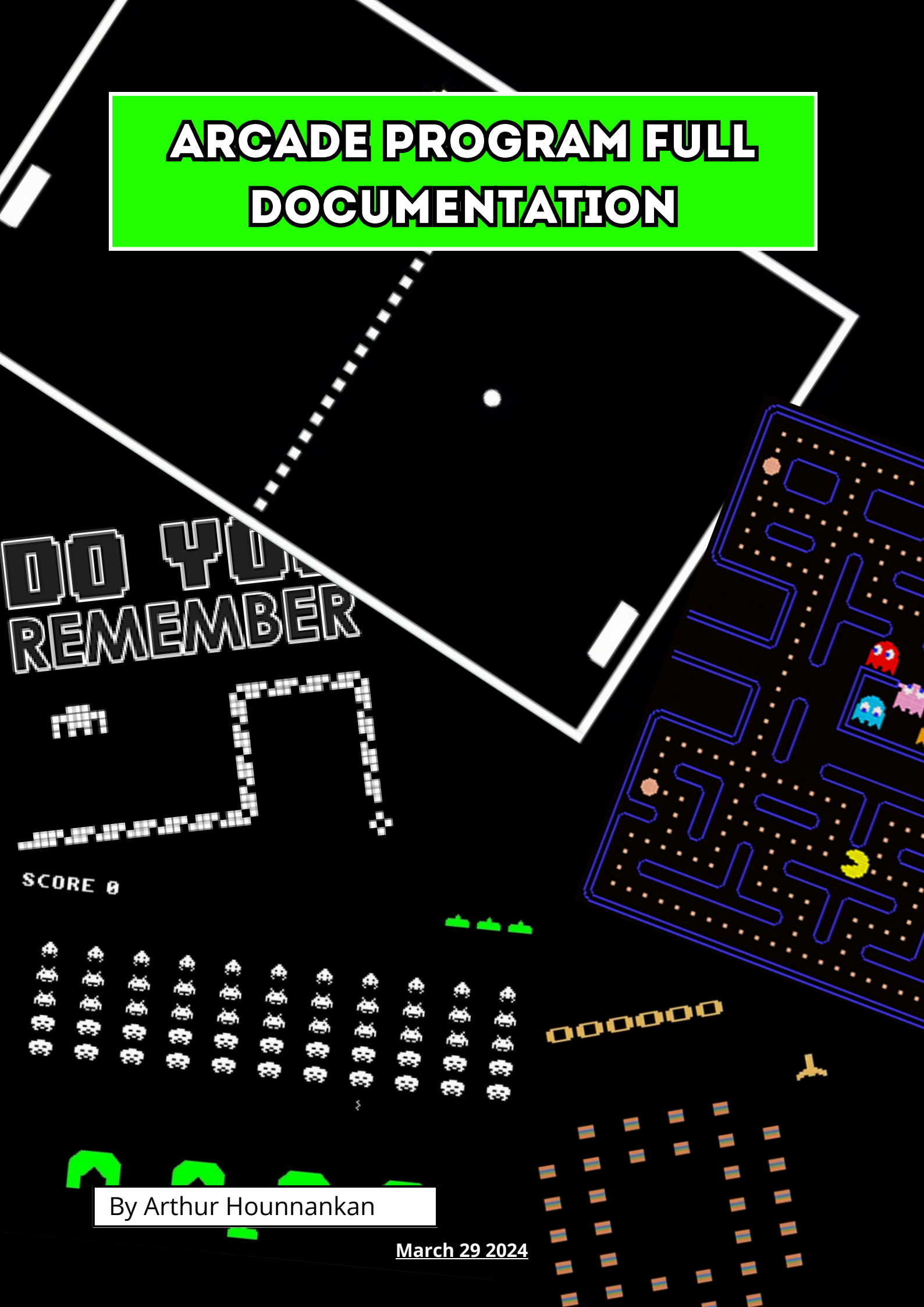
ARCADE PROGRAM FULL DOCUMENTATION



By Arthur Hounnankan

March 29 2024

Table of Content

1.Introduction

1.a Objectives and purpose of the documentation

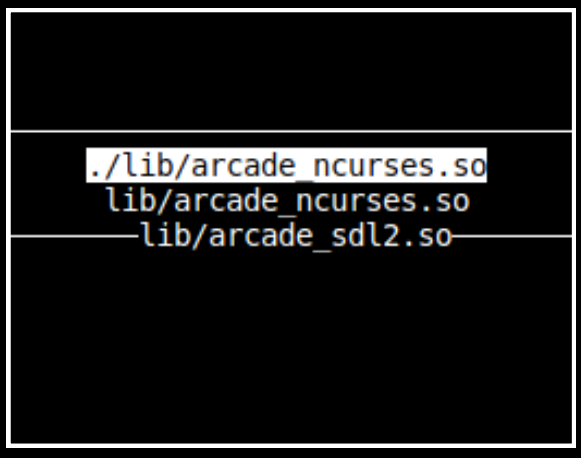
The Objectives of this documentation is to tell the reader everything there is to know about the Arcade Emulator. From the installation requirement and process to it's API and architecture.

After reading this documentation the reader should be able to create program that the emulator will be able to run. He should even be able to modify the emulator to his needs as this project is open source and allows anyone to use it even for commercial purposes.

We expect that this documentation will give the user a deep understanding of the Emulator, it's purpose, how it works, and the possibilities it offers to developers.

1.b Overview of the program

The Arcade emulator allow you to run computer version of games only available on old school consoles. Among them, **Snake**, **Solar fox**, **Pacman** and many more.
The program is able to render the games in multiples view using any library.



Ncurses view

SFML view

2.Installation Instructions

2.a System requirements

OS:
- Windows
- MacOS
- Linux

Hardware:
- Minimal: Old systems with limited resources
- Recommended: Modern hardware configuration

Dependencies:
- ncurses 6.2
- SFML 2.5.1
- SDL 2.0.14

Graphics Support:
Graphic Card supporting:
- DirectX
- OpenGL 1.1 or later

Compiler:
- g++ compiler

2.Installation Instructions

2.b **Step-by-step guide for installing the program**

Here is how you can install the program:

- Open your OS terminal
- Make sure you have installed all the dependencies (libraries, compiler, etc)
- Type: `git clone git@github.com:EpitechPromo2027/B-OOP-400-COT-4-1-arcade-arthur.hounnankan.git`
- cd B-OOP-400-COT-4-1-arcade-arthur.hounnankan/

You are now in the arcade repository.

3.Usage Guide

3.a Basic usage instructions

In the Arcade Directory you will find a **Makefile** containing a set of rules and their instructions. Here are the main on and their purpose.

core:
Compiles the core module of the program the emulator.
graphical:
Compiles the graphical modules into dynamic libraries (.so files).
games:
Compiles the games into dynamic libraries (.so files).

all:
Runs rules **core**, **graphicals** and **games**.
clean:
Cleans object files, static library files or any other files created during compilation that does not serve to run the program.
fclean:
Runs **clean** then removes the program binary.
re:
Runs **fclean** then **all**.

3.b Command-line Options

cmd: ./arcade <**path-to-initial-graphics-lib-dl-file**>

Where "path-to-initial-graphics-lib-dl-file" is the path to the .so file containing the code of the initial rendering module (library) you want to use.

Example:

```
./arcade ./lib/arcade_ncurses.so
```

4.API Documentation

4.a Description of classes, functions and methods

Namespaces:

Arcade:
This namespace contains all the classes and enumerations used by the Emulator or the core.

Implementation:

```
#ifndef ARCADE_H
    #define ARCADE_H
    #include "../../DlLoader/lib/DlLoader.hpp"
    #include "Core_OS.hpp"

namespace Arcade {
    template <typename T>
    class DlLoader;
    class Core_OS;
}
```

Arcade contains the declaration of the DlLoader module and the Core_OS class which is the class representing the emulator.

Drivers:
Drivers is just a cool name for the namespace containing the declaration of our **IGame** and **IGfx** interfaces.

Implementation:

```
#ifndef DRIVERS_H
    #define DRIVERS_H
    #include "./GAME_DISK_D/IGame.hpp"
    #include "./GFX_D/IGfx.hpp"

namespace Drivers {
    class IGame;

    class IGfx;
}

#endif /* !DRIVERS_H */
```

4.API Documentation

Classes:

Core_OS:

The **Core_OS** class represent the emulator. Two instances of class **DlLoader** one which allows use to switch graphical library and on that allows us to switch game.

Implmentation

```
#ifndef CONSOLE_OS_H
    #define CONSOLE_OS_H
    //#include "./arcade.hpp"
    #include "../../DlLoader/lib/DlLoader.hpp"
    #include "../../Drivers/Drivers.hpp"

namespace Arcade {
    class Core_OS {
        public:
            Core_OS();
            Core_OS(char *gfx_path);
            ~Core_OS();
            void run();
            void menu(std::vector<std::vector<Arcade::mapElement>> _scene, Arcade::Core_Data _data);
            void setIGfx(std::string _dl_filepath);
            void setIGame(std::string _dl_filepath);

        protected:
        private:
            Arcade::DlLoader<Drivers::IGfx> _gfxDriverLoader = Arcade::DlLoader<Drivers::IGfx>();;
            Arcade::DlLoader<Drivers::IGame> _gameDriverLoader = Arcade::DlLoader<Drivers::IGame>();;
            std::unique_ptr<Drivers::IGfx> _currentGfx;
            std::unique_ptr<Drivers::IGame> _currentGame;

            std::vector<std::vector<Arcade::mapElement>> _scene;
            Arcade::Core_Data _data;
    };
}


#endif /* !CONSOLE_OS_H */
```

Attributes:

DlLoaders

- _gfxDriverLoader
- _gameDriverLoader

Interfaces

- _currentGfx
- _currentGame

4.API Documentation

Visual & System

- _scene
- _data

_scene: A data structure containing information about all the graphical element to show on screen.
_data: Variable containing information about the user's input, the currently focused program (emulator's menu, or game)

__gfxDriverLoader: To load the graphic libraries.
_gameDriverLoader: To load the games.

_currentGfx: Unique pointer to currently used graphical library object
_currentGfx: Unique pointer to currently played game object

Methods:

- run
- menu
- setIGame
- setIGfx

_run: Contains main loop of the program
_menu: Contains the code to run the emulator's menu
_setIGame: Loads the game and sets _currentGame with it
_setIGfx: Loads the graphical lib and sets _currentGfx with it

4.API Documentation

DlLoader:

The **DlLoader** class allows you to load a dynamic lib, get symbols from it and close it when you are done using it.

Implmentation

```
#ifndef DL_LOADER_H
    #define DL_LOADER_H
    #include <iostream>
    #include <filesystem>
    #include <dlfcn.h>
    #include "../../Drivers/Drivers.hpp"
    #include "../../Core_OS/lib/Error.hpp"
    #include <memory>

namespace Arcade {
    template <typename T>
    class DlLoader {
        public:
            DlLoader();
            ~DlLoader();
            void load(std::string lib_filepath);
            std::unique_ptr<T> getInstance(std::string sym_name);
            std::string getType();
            void remove();

        protected:
        private:
            void *_dl_handle;
    };

    template class Arcade::DlLoader<Drivers::IGame>;
    template class Arcade::DlLoader<Drivers::IGfx>;
}

#define DL_DIR_PATH "./lib/"

#endif /* !DL_LOADER_H */
```

Attributes:

- _dl_handle

_dl_handle: pointer to the dynamic library handle

4.API Documentation

Methods:

- load
- getInstance
- getType
- remove

_load: loads the dl handle
_getInstance: gets an instance of a class
_getType: gets the type of a class instance
_remove: closes a dl

IGfx:

Interface of a graphical library class, contains the update method which serves as an entry-point to all graphic libs classes. 01

Implementation:

```
#ifndef IGFX_D_H
    #define IGFX_D_H
    #include <iostream>
    #include "../../Core_OS/lib/mapElement.hpp"
    #include "../../Core_OS/lib/Input.hpp"
    #include <vector>

namespace Drivers {
    class IGfx {
        public:
            virtual ~IGfx() {};
            virtual void update(std::vector<std::vector<Arcade::mapElement>> _scene, Arcade::Core_Data &_data) = 0;

        protected:
        private:
    };
}

#endif /* !IGFX_D_H */
```

Methods:

update: updates the screen display based on _scene content. Also updates player input.

4.API Documentation

IGame:
Interface of a game library class, contains the update method which serves as an entry-point to all game classes. Also contains the **getLayout** method.

Implementation:

```
#ifndef IGAME_H
    #define IGAME_H
    #include "../../Core_OS/lib/mapElement.hpp"
    #include "../../Core_OS/lib/Input.hpp"
    #include <vector>

namespace Drivers {
    class IGame {
        public:
            virtual ~IGame() {};
            virtual std::vector<std::vector<Arcade::mapElement>> getLayout() = 0;
            virtual int getScore() const = 0;
            virtual void update(Arcade::Core_Data &_data) = 0;

        protected:
        private:
    };
}

#endif /* !IGAME_H */
```

Methods:
update: updates the game state based on player player inputs and the algorithm of the game class.
getScore: gets the player score
getLayout: gets the level layout

4.API Documentation

mapElement:

Contains all information about a graphical object position, dimensions, scale, color, asset filepath, etc.

All methods in this class are to get the value of the attribute of the same name with an '_' at the beginning.

Implementation:

```hpp
#pragma once
#include <iostream>
#include "Color.hpp"

namespace Arcade {
    class mapElement
    {
        public:
            mapElement() {};
            mapElement(double width, double height, int posX, int posY, Arcade::Color color, std::string assetPath, char element) {
                _width = width;
                _height = height;
                _posX = posX;
                _posY = posY;
                _color = color;
                _assetPath = assetPath;
                _element = element;
            };
            double width() const { return _width; };
            double height() const { return _height; };
            int posX() const { return _posX; };
            int posY() const { return _posY; };
            Arcade::Color color() const { return _color; };
            std::string assetPath() const { return _assetPath; };
            char element() const { return _element; };
            ~mapElement() {};

        private:
            double _width = 0.0f;
            double _height = 0.0f;
            int _posX = 0;
            int _posY = 0;
            Arcade::Color _color;
            std::string _assetPath;
            char _element;
    };
}
```

4.API Documentation

Enumeration:

Input:

Input enumeration to tell which button was hit by the player.

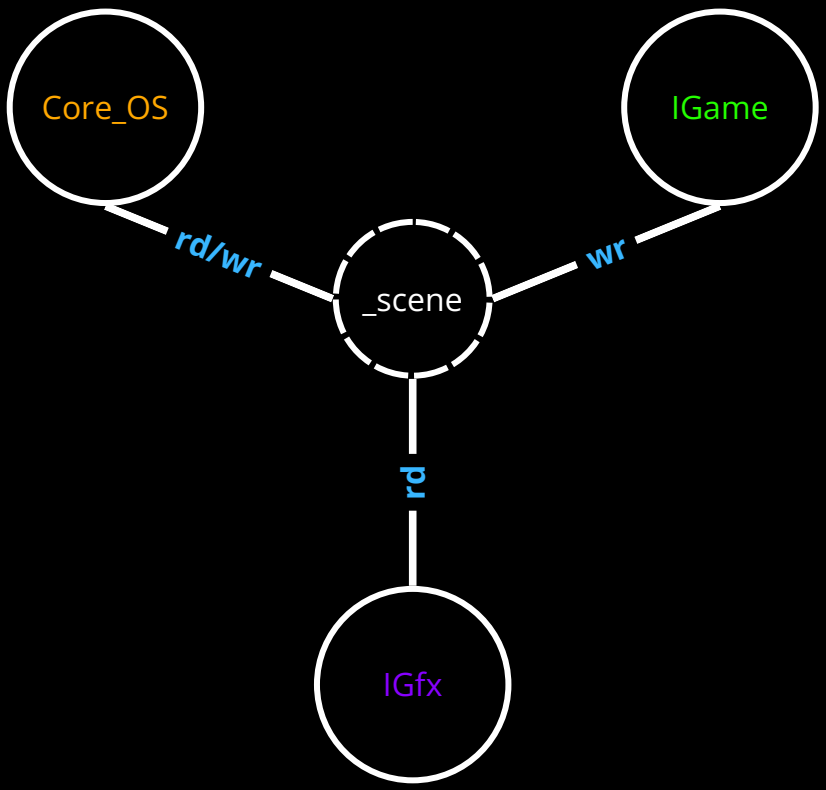
Implementation:

```
#pragma once

#include <iostream>
#include <vector>

namespace Arcade
{
    enum Input {
        UP,
        DOWN,
        LEFT,
        RIGHT,
        SPACE,
        ESCAPE,
        _none
    };
}
```

5.Architecture Overview

5.a <u>High-level overview of the program's architecture:</u>

The three main modules communicate with each other by getting and setting the content of _scene which each one of them can use.



rd: stands for read, which means the module gets the content of the _scene variable

wr: stands for write, which means the module sets the content of the _scene variable