Arcade

Generated by Doxygen 1.9.4

1 Introduction **1**

1.1 What is Arcade ? 1

1.2 What's special about it ? 1

1.3 Our arcade 2

1.4 How do you implement new games/graphics ? 2

1.5 Example: adding the game pacman 2

1.6 What's after ? 4

2 Hierarchical Index **5**

2.1 Class Hierarchy 5

3 Class Index **7**

3.1 Class List 7

4 Namespace Documentation **9**

4.1 arc Namespace Reference 9

4.1.1 Detailed Description 10

4.1.2 Enumeration Type Documentation 10

4.1.2.1 DisplayColor 10

4.1.2.2 DisplayKey 11

4.1.2.3 GameColor 12

4.1.2.4 GameKey 12

4.1.2.5 Shape 13

4.1.2.6 State 13

5 Class Documentation **15**

5.1 arc::Core Class Reference 15

5.1.1 Detailed Description 16

5.2 arc::Error Class Reference 16

5.2.1 Detailed Description 16

5.3 arc::IDisplay Class Reference 16

5.3.1 Detailed Description 17

5.4 arc::IGame Class Reference 17

5.4.1 Detailed Description 18

5.5 arc::Utils Class Reference 18

5.5.1 Detailed Description 18

Chapter 1

Introduction

1.1 What is Arcade ?

The Arcade is the 2nd project of the Object-Oriented Programming (OOP) module. It's written in C++ by 2-3 2nd year students (Dorian AYOUL and Xavier TONNELLIER). This gaming platform (should) allow you to:

- enter and save your username
- choose game and graphical dynamic libraries to play with through a starting menu
- play those games with an updated score
- create the game map you want
- be able to change graphic libraries mid-game
- save your best score (highscore)
- implement your own games and/or graphics in it

1.2 What's special about it ?

To make sure the program stays general and allows anyone to implement their own games/graphics easily, this project required the students to design its architecture by pairs of groups (at the end, one group's games should work on the other group's arcade even if they chose different graphical libraries). We did the architecture with `pierre.hamel@epitech.eu` (Pierre HAMEL and Pierre MAUGER). The architecture can be summarized to:

- the core loads the current game/graphical library
- the graphical library gets the user inputs
- the core receives them, converts them into generic enum inputs and gives them to the game library
- the game library changes the map according to the inputs
- the core receives it and, for each map tile, asks the graphical to draw the according form/letter

1.3 Our arcade

Is a buggy mess since we focused other projects/had IRL issues but the key components code is here. Here's the steps:

- Compile with 'make re' at the root
- Start with './arcade ./libs/arcade_ncurses.so' or './arcade ./libs/arcade_sfml.so' (sdl2 code is here but too unfinished)
- At any point you can press escape to quit the arcade
- Write your desired username (lowercase letters or numpad numbers) then press enter
- Select the game/graphic you want by navigating with up or down arrow then pressing enter
- You can also, from the menu or any other game, press F5/F6 for the previous/next graphic and F7/F8 for the previous/next game
- Tip: try to stay on Nibbler :)

1.4 How do you implement new games/graphics ?

To implement a new game/graphic you should:

- copy an existing game/graphic folder without its .hpp and .cpp files except entryPoint.cpp (used for loading/using/unloading the lib)
- change the names accordingly to the name of the libary you want to implement to avoid multiple definitions (Ctrl+H is useful)
- make sure to modify your folder's Makefile with the correct compilation flag(s) and library name (arcade_↩ nameofyourlib.so)
- add the rules for your folder to the general compiling Makefile in ./games or ./graphicals
- add the new library name in ./lib/libs.config under "graphicals:" or "games:"
- .cpp sourcecode files go in your folder's ./src and .hpp headers go in your folder's ./include
- make sure the classes you create are in the arc namespace and inherit from arc::IGame or arc::IDisplay (graphics)

1.5 Example: adding the game pacman

./games:

- pacman
  - **include**
    - pacman.hpp
  - **ressources**
    - map1.txt
  - **games**

    - pacman.cpp
    - entryPoint.cpp
  - Makefile

./games/pacman/include/Pacman.hpp:

```
 {c++}
#include "IGame.hpp"
namespace arc
{
    class Pacman : public arc::IGame
    {
        public:
            ~Pacman() = default;
            void init_game(void);
            void destroy_game(void);
            void update(std::vector<arc::GameKey>);
            void setGameState(arc::State state);
            arc::State getGameState(void);
            std::vector<std::vector<int» getMap(void);
            std::size_t getScore(void);
            std::string getPlayerName(void);
            std::string getGameName(void);
    };
};
```

./games/pacman/src/entryPoint.cpp:

```
 {c++}
#include "Pacman.hpp"
arc::IGame *lib = nullptr;
__attribute__((constructor))
void loadLib()
{
    lib = new arc::Pacman;
}
__attribute__((destructor))
void unloadLib()
{
    delete lib;
}
extern "C" arc::IGame *entryPoint()
{
    return lib;
}
```

./games/pacman/ressources/map1.txt:

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
x            xx            x
x xxxx xxxxx xx xxxxx xxxx x
x xxxx xxxxx xx xxxxx xxxx x
x                          x
x xxxx xx xxxxxxxx xx xxxx x
x      xx    xx    xx      x
xxxxxx xxxxx xx xxxxx xxxxxx
xxxxxx xx          xx xxxxxx
xxxxxx xx xxx  xxx xx xxxxxx
x         x      x         x
x         x      x         x
xxxxxx xx xxxxxxxx xx xxxxxx
xxxxxx xx          xx xxxxxx
xxxxxx xx xxxxxxxx xx xxxxxx
x            xx            x
x xxxx xxxxx xx xxxxx xxxx x
x   xx                xx   x
xxx xx x xxxxxxxxxx x xx xxx
x      x     xx     x      x
x xxxxxxxxxx xx xxxxxxxxxx x
x                          x
xxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

./games/pacman/Makefile:

```
SRC = src/pacman.cpp        \
      src/entryPoint.cpp    \
OBJ = $(SRC:.cpp=.o)
NAME = arcade_pacman.so
CPPFLAGS = -fno-gnu-unique -W -Wall -Wextra -fPIC -shared -rdynamic -I ./include -I ../../include
all: $(OBJ)
    g++ -o ../../lib/$(NAME) $(OBJ) $(CPPFLAGS)
clean:
    rm -rf $(OBJ)
fclean: clean
    rm -rf ../../lib/$(NAME)
```

```
re: fclean all
.PHONY: all clean fclean re
```

./games/Makefile:

```
all:
    make -C ./menu
    make -C ./pacman
clean:
    make clean -C ./menu
    make clean -C ./pacman
fclean: clean
    make fclean -C ./menu
    make fclean -C ./pacman
re: fclean all
.PHONY: all clean fclean re
```

./lib/libs.config:

```
graphicals:
arcade_ncurses.so
arcade_sdl2.so
arcade_sfml.so
games:
arcade_menu.so
arcade_pacman.so
```

Now implement your pacman in Pacman.cpp according to Pacman.hpp and you're good to go !

1.6 What's after ?

The following doxygen-generated documentation provides more infos on the different base classes in place.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

arc::Core 15
std::exception
    arc::Error 16
arc::IDisplay 16
arc::IGame 17
arc::Utils 18

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

arc::Core
Core class responsible of loading libs, transmitting player inputs to the games and transmitting the map to the graphicals 15
arc::Error
The error class safely checking for exceptions in the main 16
arc::IDisplay
The main graphical interface responsible of getting user input and drawing the map as indicated by the core 16
arc::IGame
The main graphical interface responsible of processing received user input and changing the map to be handled by the core 17
arc::Utils
A generalist utilities class that is used in different parts of the project 18

Chapter 4

Namespace Documentation

4.1 arc Namespace Reference

All classes of the project are in the arc (arcade) namespace.

Classes

- class Core

  Core class responsible of loading libs, transmitting player inputs to the games and transmitting the map to the graphicals.
- class Error

  The error class safely checking for exceptions in the main.
- class IDisplay

  The main graphical interface responsible of getting user input and drawing the map as indicated by the core.
- class IGame

  The main graphical interface responsible of processing received user input and changing the map to be handled by the core.
- class Utils

  A generalist utilities class that is used in different parts of the project.

Enumerations

- enum **DisplayColor** {
  D_RED = 1 , **D_BLUE** , **D_GREEN** , **D_WHITE** ,
  D_ORANGE , **D_CYAN** , **D_PURPLE** , **D_YELLOW** ,
  D_LIME , **D_BROWN** , **D_PINK** , **D_GRAY** ,
  D_COLOR_SIZE }
- enum **DisplayKey** {
  D_ENTER , **D_BACKSPACE** , **D_SPACE** , **D_ESCAPE** ,
  D_UP_ARROW , **D_DOWN_ARROW** , **D_LEFT_ARROW** , **D_RIGHT_ARROW** ,
  D_KEY_A , **D_KEY_B** , **D_KEY_C** , **D_KEY_D** ,
  D_KEY_E , **D_KEY_F** , **D_KEY_G** , **D_KEY_H** ,
  D_KEY_I , **D_KEY_J** , **D_KEY_K** , **D_KEY_L** ,
  D_KEY_M , **D_KEY_N** , **D_KEY_O** , **D_KEY_P** ,
  D_KEY_Q , **D_KEY_R** , **D_KEY_S** , **D_KEY_T** ,
  D_KEY_U , **D_KEY_V** , **D_KEY_W** , **D_KEY_X** ,

D_KEY_Y , **D_KEY_Z** , **D_KEY_1** , **D_KEY_2** ,
D_KEY_3 , **D_KEY_4** , **D_KEY_5** , **D_KEY_6** ,
D_KEY_7 , **D_KEY_8** , **D_KEY_9** , **D_KEY_0** ,
D_F1 , **D_F2** , **D_F3** , **D_F4** ,
D_F5 , **D_F6** , **D_F7** , **D_F8** ,
D_F9 , **D_F10** , **D_F11** , **D_F12** ,
D_KEY_SIZE }

- enum Shape { **SQUARE** = 1 , **CROSS** , **CIRCLE** }

  Shape of a form on the map to be displayed that can be associated to wall, enemy, player... (square by default)

- enum GameColor {
  G_RED = 1 , **G_BLUE** , **G_GREEN** , **G_WHITE** ,
  G_ORANGE , **G_CYAN** , **G_PURPLE** , **G_YELLOW** ,
  G_LIME , **G_BROWN** , **G_PINK** , **G_GRAY** ,
  G_COLOR_SIZE }

  The color of a map tile to display (red by default)

- enum GameKey {
  G_ENTER , **G_BACKSPACE** , **G_SPACE** , **G_ESCAPE** ,
  G_UP_ARROW , **G_DOWN_ARROW** , **G_LEFT_ARROW** , **G_RIGHT_ARROW** ,
  G_KEY_A , **G_KEY_B** , **G_KEY_C** , **G_KEY_D** ,
  G_KEY_E , **G_KEY_F** , **G_KEY_G** , **G_KEY_H** ,
  G_KEY_I , **G_KEY_J** , **G_KEY_K** , **G_KEY_L** ,
  G_KEY_M , **G_KEY_N** , **G_KEY_O** , **G_KEY_P** ,
  G_KEY_Q , **G_KEY_R** , **G_KEY_S** , **G_KEY_T** ,
  G_KEY_U , **G_KEY_V** , **G_KEY_W** , **G_KEY_X** ,
  G_KEY_Y , **G_KEY_Z** , **G_KEY_1** , **G_KEY_2** ,
  G_KEY_3 , **G_KEY_4** , **G_KEY_5** , **G_KEY_6** ,
  G_KEY_7 , **G_KEY_8** , **G_KEY_9** , **G_KEY_0** ,
  G_KEY_SIZE }

  The key inputed to be processed (can have an impact on the game)

- enum State { **STOP** = 0 , **START** = 1 , **PAUSE** = 2 }

  The current state of the game.

4.1.1 Detailed Description

All classes of the project are in the arc (arcade) namespace.

4.1.2 Enumeration Type Documentation

4.1.2.1 DisplayColor

```
enum arc::DisplayColor
```

Definition at line 15 of file IDisplay.hpp.

```
00015                       {
00016         D_RED = 1,
00017         D_BLUE,
00018         D_GREEN,
00019         D_WHITE,
00020         D_ORANGE,
00021         D_CYAN,
00022         D_PURPLE,
00023         D_YELLOW,
00024         D_LIME,
00025         D_BROWN,
00026         D_PINK,
00027         D_GRAY,
00028
00029         D_COLOR_SIZE
00030     };
```

4.1.2.2 DisplayKey

```
enum arc::DisplayKey
```

Definition at line 32 of file IDisplay.hpp.

```
00032                     {
00033         D_ENTER,
00034         D_BACKSPACE,
00035         D_SPACE,
00036         D_ESCAPE,
00037         D_UP_ARROW,
00038         D_DOWN_ARROW,
00039         D_LEFT_ARROW,
00040         D_RIGHT_ARROW,
00041         D_KEY_A,
00042         D_KEY_B,
00043         D_KEY_C,
00044         D_KEY_D,
00045         D_KEY_E,
00046         D_KEY_F,
00047         D_KEY_G,
00048         D_KEY_H,
00049         D_KEY_I,
00050         D_KEY_J,
00051         D_KEY_K,
00052         D_KEY_L,
00053         D_KEY_M,
00054         D_KEY_N,
00055         D_KEY_O,
00056         D_KEY_P,
00057         D_KEY_Q,
00058         D_KEY_R,
00059         D_KEY_S,
00060         D_KEY_T,
00061         D_KEY_U,
00062         D_KEY_V,
00063         D_KEY_W,
00064         D_KEY_X,
00065         D_KEY_Y,
00066         D_KEY_Z,
00067         D_KEY_1,
00068         D_KEY_2,
00069         D_KEY_3,
00070         D_KEY_4,
00071         D_KEY_5,
00072         D_KEY_6,
00073         D_KEY_7,
00074         D_KEY_8,
00075         D_KEY_9,
00076         D_KEY_0,
00077
00078         // Reserved to the core for changing libs/games
00079         D_F1,
00080         D_F2,
00081         D_F3,
00082         D_F4,
00083         D_F5,
00084         D_F6,
00085         D_F7,
00086         D_F8,
00087         D_F9,
00088         D_F10,
00089         D_F11,
00090         D_F12,
00091
00092         D_KEY_SIZE
00093     };
```

4.1.2.3 GameColor

`enum arc::GameColor`

The color of a map tile to display (red by default)

Definition at line 23 of file IGame.hpp.

```
00023                   {
00024         G_RED = 1,
00025         G_BLUE,
00026         G_GREEN,
00027         G_WHITE,
00028         G_ORANGE,
00029         G_CYAN,
00030         G_PURPLE,
00031         G_YELLOW,
00032         G_LIME,
00033         G_BROWN,
00034         G_PINK,
00035         G_GRAY,
00036 
00037         G_COLOR_SIZE
00038     };
```

4.1.2.4 GameKey

`enum arc::GameKey`

The key inputed to be processed (can have an impact on the game)

Definition at line 40 of file IGame.hpp.

```
00040                 {
00041         G_ENTER,
00042         G_BACKSPACE,
00043         G_SPACE,
00044         G_ESCAPE,
00045         G_UP_ARROW,
00046         G_DOWN_ARROW,
00047         G_LEFT_ARROW,
00048         G_RIGHT_ARROW,
00049         G_KEY_A,
00050         G_KEY_B,
00051         G_KEY_C,
00052         G_KEY_D,
00053         G_KEY_E,
00054         G_KEY_F,
00055         G_KEY_G,
00056         G_KEY_H,
00057         G_KEY_I,
00058         G_KEY_J,
00059         G_KEY_K,
00060         G_KEY_L,
00061         G_KEY_M,
00062         G_KEY_N,
00063         G_KEY_O,
00064         G_KEY_P,
00065         G_KEY_Q,
00066         G_KEY_R,
00067         G_KEY_S,
00068         G_KEY_T,
00069         G_KEY_U,
00070         G_KEY_V,
00071         G_KEY_W,
00072         G_KEY_X,
00073         G_KEY_Y,
00074         G_KEY_Z,
00075         G_KEY_1,
00076         G_KEY_2,
00077         G_KEY_3,
00078         G_KEY_4,
00079         G_KEY_5,
00080         G_KEY_6,
00081         G_KEY_7,
00082         G_KEY_8,
00083         G_KEY_9,
00084         G_KEY_0,
00085 
00086         G_KEY_SIZE
00087     };
```

4.1.2.5 Shape

`enum arc::Shape`

Shape of a form on the map to be displayed that can be associated to wall, enemy, player... (square by default)

Definition at line 17 of file IGame.hpp.

```
00017               {
00018         SQUARE = 1,
00019         CROSS,
00020         CIRCLE,
00021     };
```

4.1.2.6 State

`enum arc::State`

The current state of the game.

Definition at line 89 of file IGame.hpp.

```
00089               {
00090         STOP = 0,
00091         START = 1,
00092         PAUSE = 2
00093     };
```

Chapter 5

Class Documentation

5.1 arc::Core Class Reference

Core class responsible of loading libs, transmitting player inputs to the games and transmitting the map to the graphicals.

`#include <core.hpp>`

Public Member Functions

- **Core** (const std::string &path)

    Core constructor that requires the path of the starting graphical library.
- void **getLibs** ()

    Store the pathes of the game/graphic libs in ./lib/libs.config.
- void **mainLoop** ()

    Main loop of the game.
- void **loadLib** (const std::string &libPath, bool is_graph)

    Loads the given lib with dlopen/dlsym.
- void **unloadLib** (bool is_graph)

    Unloads the current used lib with dlclose.
- bool **getUsername** ()

    Gets the username at the start of the arcade.
- void **readMap** ()

    Parses the map and asks the graphic lib to draw for each tile.
- void **destroy** ()

    Destroys the game and graphic libs (when pressing escape)
- void **changeLib** ()

    Change the lib if menu.
- void **checkFunctionKey** (std::vector< DisplayKey > dKeys)

    Change the lib if function key pressed.
- void **updateKeys** (std::vector< DisplayKey > &dKeys, std::vector< GameKey > &gKeys)
- void **displayScore** ()
- void **waitClock** (std::vector< DisplayKey > &dKeys)

5.1.1 Detailed Description

Core class responsible of loading libs, transmitting player inputs to the games and transmitting the map to the graphicals.

Definition at line 19 of file core.hpp.

The documentation for this class was generated from the following file:

- core/include/core.hpp

5.2 arc::Error Class Reference

The error class safely checking for exceptions in the main.

`#include <Error.hpp>`

Inherits std::exception.

Public Member Functions

- **Error** (const std::string &error="") throw ()
  Constructor that take the error message and throws the exception to be catched.
- virtual const char ∗ **what** () const throw ()
  Conventional error message getter to display it in a catch.

5.2.1 Detailed Description

The error class safely checking for exceptions in the main.

Definition at line 16 of file Error.hpp.

The documentation for this class was generated from the following file:

- include/Error.hpp

5.3 arc::IDisplay Class Reference

The main graphical interface responsible of getting user input and drawing the map as indicated by the core.

`#include <IDisplay.hpp>`

Public Member Functions

- virtual void **initDisplay** (void)=0

  Initialize the display (creates window and sets settings)
- virtual void **destroyDisplay** (void)=0

  Destroy the display (closes window)
- virtual void **display** (void)=0

  Refresh and display the map.
- virtual void **drawSquare** (unsigned char color, std::size_t posX, std::size_t posY)=0

  Draws a square of a given color at a given position.
- virtual void **drawCircle** (unsigned char color, std::size_t posX, std::size_t posY)=0

  Draws a circle of a given color at a given position.
- virtual void **drawCross** (unsigned char color, std::size_t posX, std::size_t posY)=0

  Draws a cross of a given color at a given position.
- virtual void **drawLetter** (unsigned char letter, unsigned char color, std::size_t posX, std::size_t posY)=0

  Draws a letter of a given color at a given position.
- virtual std::vector< DisplayKey > **getKeys** (void)=0

  Get the list of player key inputs to process them.

5.3.1 Detailed Description

The main graphical interface responsible of getting user input and drawing the map as indicated by the core.

Definition at line 95 of file IDisplay.hpp.

The documentation for this class was generated from the following file:

- include/IDisplay.hpp

5.4 arc::IGame Class Reference

The main graphical interface responsible of processing received user input and changing the map to be handled by the core.

```
#include <IGame.hpp>
```

Public Member Functions

- virtual void **initGame** (void)=0

  Initializes the game (load and fill the map from .txt, set starting values)
- virtual void **destroyGame** (void)=0

  Destroy the game (delete the map)
- virtual void **update** (std::vector< GameKey > keys)=0

  Updates the game (changes the map depending on the inputs received)
- virtual void **setGameState** (State state)=0

  Sets the state of the game to running, stopped (menu) or paused.
- virtual State **getGameState** (void)=0

  Gets the stats of the game.

- virtual std::vector< std::vector< int > > **getMap** (void)=0
  Gets the map of the game to manipulate it.
- virtual std::size_t **getScore** (void)=0
  Gets the player score.
- virtual std::string **getPlayerName** (void)=0
  Gets the player name.
- virtual void **setPlayerName** (std::string)=0
  Sets the game name.
- virtual std::string **getGameName** (void)=0
  Gets the game name.

5.4.1 Detailed Description

The main graphical interface responsible of processing received user input and changing the map to be handled by the core.

Definition at line 95 of file IGame.hpp.

The documentation for this class was generated from the following file:

- include/IGame.hpp

5.5 arc::Utils Class Reference

A generalist utilities class that is used in different parts of the project.

```
#include <Utils.hpp>
```

Public Member Functions

- std::vector< std::pair< int, int > > **generateRand** (std::vector< std::vector< int > > map, size_t number, std::vector< int > obs)
  Generates a random position.
- std::vector< std::vector< int > > **convertMap** (std::vector< std::vector< int > > map)
  Converts the map into bitshifted characters containing the form + color.

5.5.1 Detailed Description

A generalist utilities class that is used in different parts of the project.

Definition at line 17 of file Utils.hpp.

The documentation for this class was generated from the following file:

- include/Utils.hpp