The Condemned Ship

2.0

Generated by Doxygen 1.8.14

Contents

Chapter 1

The Condemned Ship: an Adventure

1.1 Introduction

This project was started by a group of five students of Computer Science (the group **FSC**) as part of an exam and it developed into something bigger than it originally was.

1.2 This website

This website contains the documentation of both the code and the project itself. It has been generated using Doxygen and stylized using the M.CSS's Doxygen template.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 colored_string Class Reference

A colored string This class allows to print a colored string to the stdout. This is a cross platform solution.

`#include <utilities.h>`

Public Types

- enum PrintColors : short {
  PrintColors::BLACK = 0, PrintColors::RED = 1, PrintColors::GREEN = 2, PrintColors::YELLOW = 3,
  PrintColors::BLUE = 4, PrintColors::MAGENTA = 5, PrintColors::CYAN = 6, PrintColors::WHITE = 7 }

  Printable colors. This enumerator contains all the possible colors that can be printed. Each color is identified by an integer.

Public Member Functions

- colored_string (const std::string &pToPrint, const PrintColors pForeground=PrintColors::RED, const PrintColors pBackground=PrintColors::BLACK)

  Colored string's constructor. This creates a new colored string ready to be printed.

Private Attributes

- std::string **str**
- std::string **color**

Friends

- std::ostream & operator<< (std::ostream &os, const colored_string &str)

  Output the colored string. This outputs the colored string to a std::ostream (like `cout`).

4.1.1 Detailed Description

A colored string This class allows to print a colored string to the stdout. This is a cross platform solution.

4.1.2 Member Enumeration Documentation

4.1.2.1 PrintColors

```
enum colored_string::PrintColors : short [strong]
```

Printable colors. This enumerator contains all the possible colors that can be printed. Each color is identified by an integer.

Warning

The colors have various code based on the platform. This is because some operative systems (like Windows) don't support ASCII Escaped sequences.

Enumerator

BLACK	The black color.
RED	The red color.
GREEN	The green color.
YELLOW	The yellow color.
BLUE	The blue color.
MAGENTA	The magenta color.
CYAN	The cyan color.
WHITE	The white color.

4.1.3 Constructor & Destructor Documentation

4.1.3.1 colored_string()

```
colored_string::colored_string (
            const std::string & pToPrint,
            const PrintColors pForeground = PrintColors::RED,
            const PrintColors pBackground = PrintColors::BLACK )
```

Colored string's constructor. This creates a new colored string ready to be printed.

Parameters

in	*pToPrint*	The normal string.
in	*pForeground*	The foreground color (default is PrintColors::RED).
in	*pBackground*	The background color (default is PrintColors::BLACK).

4.1.4 Friends And Related Function Documentation

4.1.4.1 operator<<

```
std::ostream& operator<< (
            std::ostream & os,
            const colored_string & str )  [friend]
```

Output the colored string. This outputs the colored string to a std::ostream (like `cout`).

Note

This function does *not* add a new line at the end of the output.

Parameters

`in`	*os*	The output stream.
`in`	*str*	The colored string.

Returns

The modified output stream.

Example

The following line of code will print `"Hello World!"` (with an ending new line character) to the stdout. The string will be printed in *red*, using the color *blue* as background.

```
std::cout << colored_string("Hello world!",
                 colored_string::PrintColors::RED,
                 colored_string::PrintColors::BLUE)
          << std::endl;
```

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

- utilities.h

4.2 game Class Reference

The game. This class contains all the core functions of the game. It's in this class that the main loop of the game can be found.

`#include <game.h>`

Classes

- class languages

  An array of languages. This class is a wrapper for an array of languages. It is used to check data types and avoid errors.

Public Member Functions

- void **begin** ()

Protected Types

- enum String_Resources : unsigned {
  GAME_TITLE = 0, ORIGINAL_AUTHOR, AUTHOR, COPYRIGHT,
  VERSION, INTRODUCTION, ERROR_STRING, MENU_FIRST_OPTION,
  MENU_SECOND_OPTION, MENU_EXIT, MENU_INPUT_PROMPT, LANGUAGE_SUBMENU_TITLE,
  RES_STRING_NUMBER }

  All the game's strings' codes. This enumerator is used to get the code associated with a string. This allows to write a more readable code.

Protected Member Functions

- void exec ()

  The main loop. This is the main loop of the game. In this loop, all the user's input (regarding actions in the game) and game events take place.

Protected Attributes

- std::string mStrings [RES_STRING_NUMBER]

  The array containing all the game strings.
- languages mLanguages = languages({{"it", "Italiano"}, {"en", "English"}})

  The object containing the array of languages.
- languages::AvailableLanguages mCurrentLang

  The current selected language's code.
- bool mEndGame

  Does the game have to end?

Private Member Functions

- void **end_game** ()
- void **change_language** ()
- unsigned **show_menu** ()
- void **show_intro** ()
- void **get_strings** ()

4.2.1 Detailed Description

The game. This class contains all the core functions of the game. It's in this class that the main loop of the game can be found.

4.2.2 Member Enumeration Documentation

4.2.2.1 String_Resources

```
enum game::String_Resources : unsigned [protected]
```

All the game's strings' codes. This enumerator is used to get the code associated with a string. This allows to write a more readable code.

Warning

Do *not* modify the first and last values: this enumerator is used to index an array.

Enumerator

GAME_TITLE	The game's title.
ORIGINAL_AUTHOR	The original game's authors.
AUTHOR	The modified game's authors.
COPYRIGHT	A copyright notice.
VERSION	The game's version.
INTRODUCTION	The game's introduction.
ERROR_STRING	The error message that will be printed if an input fails.
MENU_FIRST_OPTION	The first option of the menu.
MENU_SECOND_OPTION	The second option of the menu.
MENU_EXIT	The "Exit" option of the menu.
MENU_INPUT_PROMPT	The message that will be printed to wait a user input in the menu.
LANGUAGE_SUBMENU_TITLE	The title of the language selection sub-menu.
RES_STRING_NUMBER	A useful constant that indicates how many strings are being saved.

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

- game.h
- game.cpp

4.3 game::languages::language Struct Reference

a language

```
#include <game.h>
```

Public Attributes

- std::string ISO639_1
    The ISO 639-1 code of the language (two letters code).
- std::string name
    The name of the language. This is a name that can be printed and selected by the user.

4.3.1 Detailed Description

a language

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

- game.h

4.4 game::languages Class Reference

An array of languages. This class is a wrapper for an array of languages. It is used to check data types and avoid errors.

```
#include <game.h>
```

Classes

- struct language

  a language

Public Types

- enum AvailableLanguages : unsigned { ITALIAN = 0, ENGLISH, NUMBER_OF_AVAILABLE_LANGUAGES }

  All the available languages. This enumerator is used to get a particular language by a costant and is useful to make the code more readable.

Public Member Functions

- languages (const language(&pLang)[NUMBER_OF_AVAILABLE_LANGUAGES]) noexcept

  Languages' array's constructor. This construct a new languages' array.
- language & operator[ ] (AvailableLanguages lang) noexcept

  Get a language. This gets a language using its code.

Private Attributes

- language mLanguages [NUMBER_OF_AVAILABLE_LANGUAGES]

  The underlaying array of languages.

4.4.1 Detailed Description

An array of languages. This class is a wrapper for an array of languages. It is used to check data types and avoid errors.

4.4.2 Member Enumeration Documentation

4.4.2.1 AvailableLanguages

```
enum game::languages::AvailableLanguages : unsigned
```

All the available languages. This enumerator is used to get a particular language by a costant and is useful to make the code more readable.

Warning

Do *not* modify the first and last values: this enumerator is used to index an array.

Enumerator

ITALIAN	The constant for the *Italian* language.
ENGLISH	The constant for the *English* language.
NUMBER_OF_AVAILABLE_LANGUAGES	A useful constant that indicates how many languages are available.

4.4.3 Constructor & Destructor Documentation

4.4.3.1 languages()

```
game::languages::languages (
            const language(&) pLang[NUMBER_OF_AVAILABLE_LANGUAGES] )  [noexcept]
```

Languages' array's constructor. This construct a new languages' array.

Parameters

in	*pLang*	An array of languages.

4.4.4 Member Function Documentation

4.4.4.1 operator[]()

```
game::languages::language & game::languages::operator[] (
            AvailableLanguages lang )  [noexcept]
```

Get a language. This gets a language using its code.

Parameters

`in`	*lang*	The language's code. It must be one defined in the AvailableLanguages enumerator.

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

- game.h
- game.cpp

Chapter 5

File Documentation

5.1 game.h File Reference

The main header of the project. This header contains the declaration of all the core functions of the game.

```
#include <string>
#include <array>
```

Classes

- class game

  The game. This class contains all the core functions of the game. It's in this class that the main loop of the game can be found.
- class game::languages

  An array of languages. This class is a wrapper for an array of languages. It is used to check data types and avoid errors.
- struct game::languages::language

  a language

5.1.1 Detailed Description

The main header of the project. This header contains the declaration of all the core functions of the game.

Copyright

GNU General Public License version 3.

5.2 utilities.h File Reference

Some utilities functions. This file contains the declaration and definition of various miscellaneous functions that are useful in various parts of the project.

```
#include <iostream>
#include <string>
#include <math.h>
#include <limits>
```

Classes

- class colored_string

  A colored string This class allows to print a colored string to the stdout. This is a cross platform solution.

Functions

- template<class InputType >
  InputType **get_value_in_range** (const InputType &min, const InputType &max, const std::string &pInput↩
  Prompt, const std::string &pErrorPrompt)
- void **clear_screen** ()
- void **press_any_key** ()
- std::string center_string (const std::string &s, unsigned width=80u)

  Center a string. Given a string, this functions centers it with spaces.
- std::ostream & operator<< (std::ostream &os, const colored_string &str)

5.2.1 Detailed Description

Some utilities functions. This file contains the declaration and definition of various miscellaneous functions that are useful in various parts of the project.

Copyright

GNU General Public License version 3.

5.2.2 Function Documentation

5.2.2.1 center_string()

```
std::string center_string (
             const std::string & s,
             unsigned width = 80u )
```

Center a string. Given a string, this functions centers it with spaces.

Parameters

in	*s*	The string to be centered.
in	*width*	The total width of the final string. This is the total width on which the string has to be centered.

Returns

The centered string with trailing spaces (both at the beginning and the ending).

5.2.2.2 operator<<()

```
std::ostream& operator<< (
             std::ostream & os,
             const colored_string & str )
```

Note

This function does *not* add a new line at the end of the output.

Parameters

in	*os*	The output stream.
in	*str*	The colored string.

Returns

The modified output stream.

Example

The following line of code will print `"Hello World!"` (with an ending new line character) to the stdout. The string will be printed in *red*, using the color *blue* as background.

```
std::cout << colored_string("Hello world!",
                 colored_string::PrintColors::RED,
                 colored_string::PrintColors::BLUE)
          << std::endl;
```

Index