HelloWorld

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 InputsHandler Class Reference

A class which handles inputs from peripherals (i.e., keyboard and mouse).

```
#include <InputsHandler.h>
```

Public Member Functions

- InputsHandler (void)

  Constructor for the InputsHandler class.
- void process (sf::Event ∗)
- void printKeysPressed (void)

  Method to print out which keys are currently pressed.
- void reset (void)

  Method to reset InputsHandler. To be called once per frame (at end of frame!).
- ∼InputsHandler (void)

  Destructor for the InputsHandler class.

Public Attributes

- std::vector< bool > key_pressed_once_vec
- std::vector< bool > key_press_vec
- std::map< sf::Keyboard::Key, std::string > key_code_map

Private Member Functions

- void __constructKeyCodeMap (void)

  Helper method to construct a map from sf::Keyboard::Key to a string representation of the corresponding key.

3.1.1 Detailed Description

A class which handles inputs from peripherals (i.e., keyboard and mouse).

3.1.2 Constructor & Destructor Documentation

3.1.2.1 InputsHandler()

```
InputsHandler::InputsHandler (
              void  )
```

Constructor for the InputsHandler class.

```
{
    this->key_pressed_once_vec.resize(sf::Keyboard::KeyCount, false);
    this->key_press_vec.resize(sf::Keyboard::KeyCount, false);

    this->__constructKeyCodeMap();

    std::cout « "InputsHandler constructed at " « this « std::endl;

    return;
}   /* InputsHandler() */
```

3.1.2.2 ∼InputsHandler()

```
InputsHandler::∼InputsHandler (
              void  )
```

Destructor for the InputsHandler class.

```
{
    std::cout « "InputsHandler at " « this « " destroyed" « std::endl;

    return;
}   /* ~InputsHandler() */
```

3.1.3 Member Function Documentation

3.1.3.1 __constructKeyCodeMap()

```
void InputsHandler::__constructKeyCodeMap (
              void  )  [private]
```

Helper method to construct a map from sf::Keyboard::Key to a string representation of the corresponding key.

```
{
    //  1. unknown keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Unknown, "Unknown")
    );


    //  2. alpha keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::A, "A")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::B, "B")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::C, "C")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::D, "D")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::E, "E")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F, "F")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::G, "G")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::H, "H")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::I, "I")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::J, "J")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::K, "K")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::L, "L")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::M, "M")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::N, "N")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::O, "O")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::P, "P")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Q, "Q")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::R, "R")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::S, "S")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::T, "T")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::U, "U")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::V, "V")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::W, "W")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::X, "X")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Y, "Y")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Z, "Z")
    );


    //  3. numeric keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num0, "0")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num1, "1")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num2, "2")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num3, "3")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num4, "4")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num5, "5")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num6, "6")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num7, "7")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num8, "8")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Num9, "9")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad0, "0")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad1, "1")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad2, "2")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad3, "3")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad4, "4")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad5, "5")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad6, "6")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad7, "7")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad8, "8")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Numpad9, "9")
    );


    //  4. direction keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Left, "Left")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Right, "Right")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Up, "Up")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Down, "Down")
    );


    //  5. function keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F1, "F1")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F2, "F2")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F3, "F3")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F4, "F4")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F5, "F5")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F6, "F6")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F7, "F7")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F8, "F8")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F9, "F9")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F10, "F10")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F11, "F11")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F12, "F12")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F13, "F13")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F14, "F14")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::F15, "F15")
    );


    //  6. other keys
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Escape, "Escape")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LControl, "LCtrl")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LShift, "LShift")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LAlt, "LAlt")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LSystem, "LSystem")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RControl, "RCtrl")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RShift, "RShift")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RAlt, "RAlt")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RSystem, "RSystem")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Menu, "Menu")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::LBracket, "LBracket")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::RBracket, "RBracket")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Semicolon, "Semicolon")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Comma, "Comma")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Period, "Period")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Quote, "Quote")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Slash, "Slash")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Backslash, "Backslash")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Tilde, "Tilde")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Equal, "Equal")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Hyphen, "Hyphen")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Space, "Space")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Enter, "Enter")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Backspace, "Backspace")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Tab, "Tab")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::PageUp, "PageUp")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::PageDown, "PageDown")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::End, "End")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Home, "Home")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Insert, "Insert")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Delete, "Delete")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Add, "Add")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Subtract, "Subtract")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Multiply, "Multiply")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Divide, "Divide")
    );
    this->key_code_map.insert(
        std::pair<sf::Keyboard::Key, std::string>(sf::Keyboard::Pause, "Pause")
    );

    return;
}   /* __constructKeyCodeMap() */
```

3.1.3.2 printKeysPressed()

```
void InputsHandler::printKeysPressed (
            void   )
```

Method to print out which keys are currently pressed.

```
{
    std::string print_str = "";

    for (size_t i = 0; i < this->key_press_vec.size(); i++) {
        if (this->key_press_vec[i]) {
            print_str += this->key_code_map[sf::Keyboard::Key(i)];
            print_str += ", ";
        }
    }

    if (not print_str.empty()) {
        std::cout « "Keys pressed: " « print_str « std::endl;
    }

    return;
}   /* printKeysPressed() */
```

3.1.3.3 process()

```
void InputsHandler::process (
            sf::Event * event_ptr )
```

```
{
    //  1. update state of key press vectors
    switch (event_ptr->type) {
        case (sf::Event::KeyPressed): {
            if (not this->key_press_vec[event_ptr->key.code]) {
                this->key_pressed_once_vec[event_ptr->key.code] = true;
            }

            this->key_press_vec[event_ptr->key.code] = true;

            break;
        }

        case (sf::Event::KeyReleased): {
            this->key_pressed_once_vec[event_ptr->key.code] = false;
            this->key_press_vec[event_ptr->key.code] = false;

            break;
        }

        default: {
            //  do nothing!

            break;
        }
    }

    return;
}   /* process() */
```

3.1.3.4 reset()

```
void InputsHandler::reset (
            void  )
```

Method to reset InputsHandler. To be called once per frame (at end of frame!).

```
{
    for (size_t i = 0; i < this->key_press_vec.size(); i++) {
        this->key_pressed_once_vec[i] = false;
    }

    return;
}   /* reset() */
```

3.1.4 Member Data Documentation

3.1.4.1 key_code_map

```
std::map<sf::Keyboard::Key, std::string> InputsHandler::key_code_map
```

3.1.4.2 key_press_vec

```
std::vector<bool> InputsHandler::key_press_vec
```

3.1.4.3 key_pressed_once_vec

```
std::vector<bool> InputsHandler::key_pressed_once_vec
```

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

- header/ESC_core/InputsHandler.h
- source/ESC_core/InputsHandler.cpp

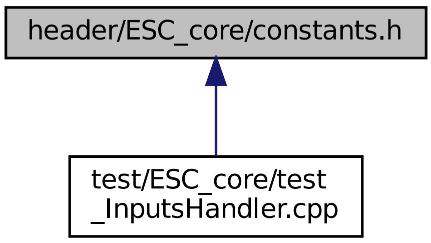
Chapter 4

File Documentation

4.1 header/ESC_core/constants.h File Reference

Header file for various constants.

This graph shows which files directly or indirectly include this file:


header/ESC_core/constants.h
test/ESC_core/test_InputsHandler.cpp

Variables

- const int FRAMES_PER_SECOND = 60
- const double SECONDS_PER_FRAME = 1.0 / 60

4.1.1 Detailed Description

Header file for various constants.

4.1.2 Variable Documentation

4.1.2.1 FRAMES_PER_SECOND

```
const int FRAMES_PER_SECOND = 60
```

4.1.2.2 SECONDS_PER_FRAME

```
const double SECONDS_PER_FRAME = 1.0 / 60
```

4.2 header/ESC_core/doxygen_cite.h File Reference

Header file which simply cites the doxygen tool.

4.2.1 Detailed Description

Header file which simply cites the doxygen tool.

Ref: van Heesch. [2023]

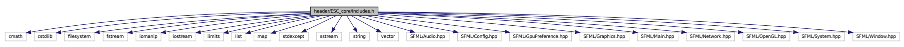
4.3 header/ESC_core/includes.h File Reference

Header file for various includes.

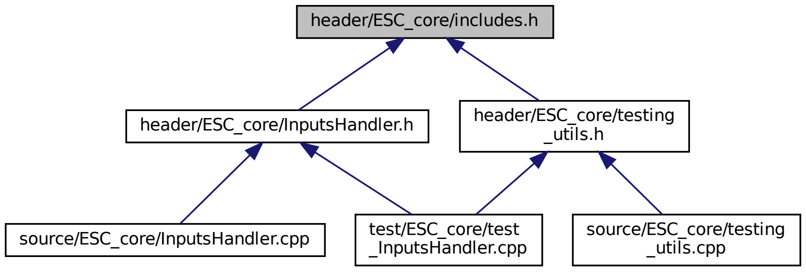
```
#include <cmath>
#include <cstdlib>
#include <filesystem>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <limits>
#include <list>
#include <map>
#include <stdexcept>
#include <sstream>
#include <string>
#include <vector>
#include <SFML/Audio.hpp>
#include <SFML/Config.hpp>
#include <SFML/GpuPreference.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Main.hpp>
#include <SFML/Network.hpp>
#include <SFML/OpenGL.hpp>
#include <SFML/System.hpp>
```

```
#include <SFML/Window.hpp>
```

Include dependency graph for includes.h:



This graph shows which files directly or indirectly include this file:


header/ESC_core/includes.h
header/ESC_core/InputsHandler.h
header/ESC_core/testing_utils.h
source/ESC_core/InputsHandler.cpp
test/ESC_core/test_InputsHandler.cpp
source/ESC_core/testing_utils.cpp

4.3.1 Detailed Description

Header file for various includes.

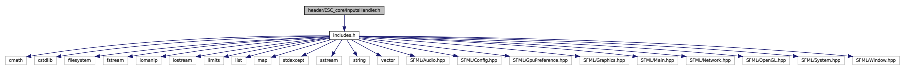
Ref: Gomila [2023]

4.4 header/ESC_core/InputsHandler.h File Reference

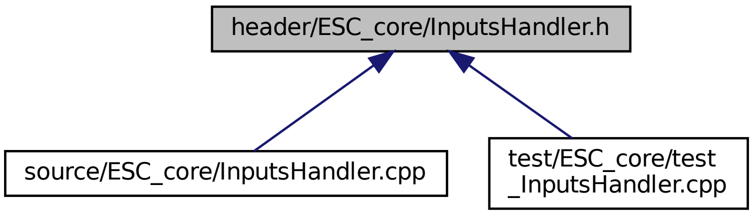
Header file for the InputsHandler class.

```
#include "includes.h"
```

Include dependency graph for InputsHandler.h:



This graph shows which files directly or indirectly include this file:


header/ESC_core/InputsHandler.h
source/ESC_core/InputsHandler.cpp
test/ESC_core/test
_InputsHandler.cpp

Classes

- class InputsHandler

  A class which handles inputs from peripherals (i.e., keyboard and mouse).

4.4.1 Detailed Description

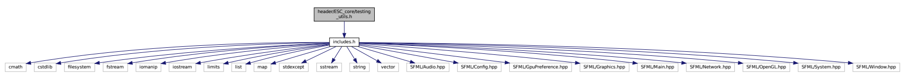
Header file for the InputsHandler class.

4.5 header/ESC_core/testing_utils.h File Reference

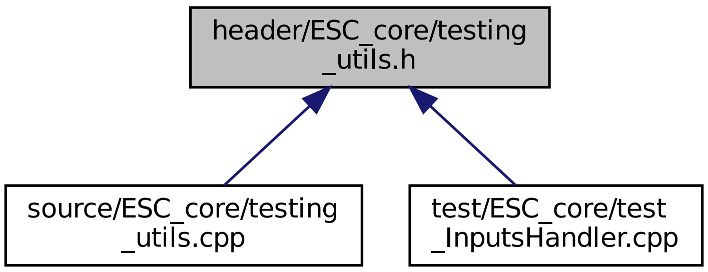
Header file for various testing utilities.

```
#include "includes.h"
```

Include dependency graph for testing_utils.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define FLOAT_TOLERANCE 1e-6

  A tolerance for application to floating point equality tests.

Functions

- void printGreen (std::string)

  A function that sends green text to std::cout.
- void printGold (std::string)

  A function that sends gold text to std::cout.
- void printRed (std::string)

  A function that sends red text to std::cout.
- void testFloatEquals (double, double, std::string, int)

  Tests for the equality of two floating point numbers x and y (to within FLOAT_TOLERANCE).
- void testGreaterThan (double, double, std::string, int)

  Tests if $x > y$.
- void testGreaterThanOrEqualTo (double, double, std::string, int)

  Tests if $x >= y$.
- void testLessThan (double, double, std::string, int)

  Tests if $x < y$.
- void testLessThanOrEqualTo (double, double, std::string, int)

  Tests if $x <= y$.
- void testTruth (bool, std::string, int)

  Tests if the given statement is true.
- void expectedErrorNotDetected (std::string, int)

  A utility function to print out a meaningful error message whenever an expected error fails to be thrown/caught/detected.

4.5.1 Detailed Description

Header file for various testing utilities.

This is a library of utility functions used throughout the various test suites.

4.5.2 Macro Definition Documentation

4.5.2.1 FLOAT_TOLERANCE

```
#define FLOAT_TOLERANCE 1e-6
```

A tolerance for application to floating point equality tests.

4.5.3 Function Documentation

4.5.3.1 expectedErrorNotDetected()

```
void expectedErrorNotDetected (
            std::string file,
            int line )
```

A utility function to print out a meaningful error message whenever an expected error fails to be thrown/caught/detected.

Parameters

file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    std::string error_str = "\n ERROR  failed to throw expected error prior to line ";
    error_str += std::to_string(line);
    error_str += " of ";
    error_str += file;

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* expectedErrorNotDetected() */
```

4.5.3.2 printGold()

```
void printGold (
              std::string input_str )
```

A function that sends gold text to std::cout.

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[33m" « input_str « "\033[0m";
    return;
}   /* printGold() */
```

4.5.3.3 printGreen()

```
void printGreen (
              std::string input_str )
```

A function that sends green text to std::cout.

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[32m" « input_str « "\033[0m";
    return;
}   /* printGreen() */
```

4.5.3.4 printRed()

```
void printRed (
            std::string input_str )
```

A function that sends red text to std::cout.

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[31m" « input_str « "\033[0m";
    return;
}   /* printRed() */
```

4.5.3.5 testFloatEquals()

```
void testFloatEquals (
            double x,
            double y,
            std::string file,
            int line )
```

Tests for the equality of two floating point numbers *x* and *y* (to within FLOAT_TOLERANCE).

Parameters

x	The first of two numbers to test.
y	The second of two numbers to test.
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (fabs(x - y) <= FLOAT_TOLERANCE) {
        return;
    }

    std::string error_str = "ERROR: testFloatEquals():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " and ";
    error_str += std::to_string(y);
    error_str += " are not equal to within +/- ";
    error_str += std::to_string(FLOAT_TOLERANCE);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testFloatEquals() */
```

4.5.3.6 testGreaterThan()

```
void testGreaterThan (
            double x,
            double y,
            std::string file,
            int line )
```

Tests if x > y.

Parameters

x	The first of two numbers to test.
y	The second of two numbers to test.
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (x > y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThan() */
```

4.5.3.7 testGreaterThanOrEqualTo()

```
void testGreaterThanOrEqualTo (
                double x,
                double y,
                std::string file,
                int line )
```

Tests if x >= y.

Parameters

x	The first of two numbers to test.
y	The second of two numbers to test.
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (x >= y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThanOrEqualTo() */
```

4.5.3.8 testLessThan()

```
void testLessThan (
            double x,
            double y,
            std::string file,
            int line )
```

Tests if x < y.

Parameters

x	The first of two numbers to test.
y	The second of two numbers to test.
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (x < y) {
        return;
    }

    std::string error_str = "ERROR: testLessThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThan() */
```

4.5.3.9 testLessThanOrEqualTo()

```
void testLessThanOrEqualTo (
            double x,
            double y,
            std::string file,
            int line )
```

Tests if x <= y.

Parameters

x	The first of two numbers to test.
y	The second of two numbers to test.
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (x <= y) {
        return;
    }

    std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThanOrEqualTo() */
```

4.5.3.10 testTruth()

```
void testTruth (
            bool statement,
            std::string file,
            int line )
```

Tests if the given statement is true.

Parameters

statement	The statement whose truth is to be tested ("1 == 0", for example).
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

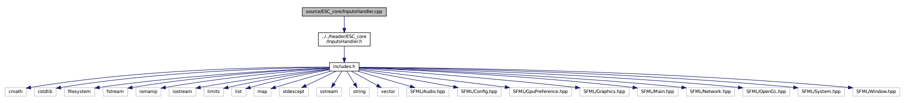
```
{
    if (statement) {
        return;
    }

    std::string error_str = "ERROR: testTruth():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += "Given statement is not true";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testTruth() */
```

4.6 source/ESC_core/InputsHandler.cpp File Reference

Implementation file for the InputsHandler class.

```
#include "../../header/ESC_core/InputsHandler.h"
```

Include dependency graph for InputsHandler.cpp:



4.6.1 Detailed Description

Implementation file for the InputsHandler class.

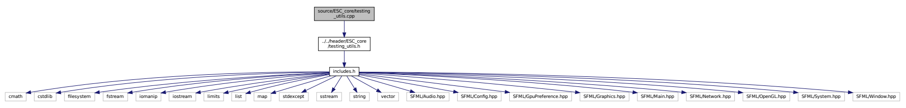
A class which handles inputs from peripherals (i.e., keyboard and mouse).

4.7 source/ESC_core/testing_utils.cpp File Reference

Implementation file for various testing utilities.

```
#include "../../header/ESC_core/testing_utils.h"
```

Include dependency graph for testing_utils.cpp:



Functions

- void printGreen (std::string input_str)

  A function that sends green text to std::cout.
- void printGold (std::string input_str)

  A function that sends gold text to std::cout.
- void printRed (std::string input_str)

  A function that sends red text to std::cout.
- void testFloatEquals (double x, double y, std::string file, int line)

  Tests for the equality of two floating point numbers x and y (to within FLOAT_TOLERANCE).
- void testGreaterThan (double x, double y, std::string file, int line)

  Tests if $x > y$.
- void testGreaterThanOrEqualTo (double x, double y, std::string file, int line)

  Tests if $x >= y$.
- void testLessThan (double x, double y, std::string file, int line)

  Tests if $x < y$.
- void testLessThanOrEqualTo (double x, double y, std::string file, int line)

  Tests if $x <= y$.
- void testTruth (bool statement, std::string file, int line)

  Tests if the given statement is true.
- void expectedErrorNotDetected (std::string file, int line)

  A utility function to print out a meaningful error message whenever an expected error fails to be thrown/caught/detected.

4.7.1 Detailed Description

Implementation file for various testing utilities.

This is a library of utility functions used throughout the various test suites.

4.7.2 Function Documentation

4.7.2.1 expectedErrorNotDetected()

```
void expectedErrorNotDetected (
             std::string file,
             int line )
```

A utility function to print out a meaningful error message whenever an expected error fails to be thrown/caught/detected.

Parameters

file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    std::string error_str = "\n ERROR  failed to throw expected error prior to line ";
    error_str += std::to_string(line);
    error_str += " of ";
    error_str += file;

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* expectedErrorNotDetected() */
```

4.7.2.2 printGold()

```
void printGold (
             std::string input_str )
```

A function that sends gold text to std::cout.

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[33m" « input_str « "\033[0m";
    return;
}   /* printGold() */
```

4.7.2.3 printGreen()

```
void printGreen (
             std::string input_str )
```

A function that sends green text to std::cout.

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[32m" « input_str « "\033[0m";
    return;
}   /* printGreen() */
```

4.7.2.4 printRed()

```
void printRed (
             std::string input_str )
```

A function that sends red text to std::cout.

Parameters

input_str	The text of the string to be sent to std::cout.

```
{
    std::cout « "\x1B[31m" « input_str « "\033[0m";
    return;
}   /* printRed() */
```

4.7.2.5 testFloatEquals()

```
void testFloatEquals (
             double x,
             double y,
             std::string file,
             int line )
```

Tests for the equality of two floating point numbers *x* and *y* (to within FLOAT_TOLERANCE).

Parameters

x	The first of two numbers to test.
y	The second of two numbers to test.
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (fabs(x - y) <= FLOAT_TOLERANCE) {
        return;
    }

    std::string error_str = "ERROR: testFloatEquals():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " and ";
    error_str += std::to_string(y);
    error_str += " are not equal to within +/- ";
    error_str += std::to_string(FLOAT_TOLERANCE);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testFloatEquals() */
```

4.7.2.6 testGreaterThan()

```
void testGreaterThan (
              double x,
              double y,
              std::string file,
              int line )
```

Tests if x > y.

Parameters

x	The first of two numbers to test.
y	The second of two numbers to test.
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (x > y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThan() */
```

4.7.2.7 testGreaterThanOrEqualTo()

```
void testGreaterThanOrEqualTo (
              double x,
              double y,
              std::string file,
              int line )
```

Tests if x >= y.

Parameters

x	The first of two numbers to test.
y	The second of two numbers to test.
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (x >= y) {
        return;
    }

    std::string error_str = "ERROR: testGreaterThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not greater than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testGreaterThanOrEqualTo() */
```

4.7.2.8 testLessThan()

```
void testLessThan (
            double x,
            double y,
            std::string file,
            int line )
```

Tests if x < y.

Parameters

x	The first of two numbers to test.
y	The second of two numbers to test.
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (x < y) {
        return;
    }

    std::string error_str = "ERROR: testLessThan():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThan() */
```

4.7.2.9 testLessThanOrEqualTo()

```
void testLessThanOrEqualTo (
            double x,
            double y,
            std::string file,
            int line )
```

Tests if x <= y.

Parameters

x	The first of two numbers to test.
y	The second of two numbers to test.
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

```
{
    if (x <= y) {
        return;
    }

    std::string error_str = "ERROR: testLessThanOrEqualTo():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += std::to_string(x);
    error_str += " is not less than or equal to ";
    error_str += std::to_string(y);
    error_str += "\n";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testLessThanOrEqualTo() */
```

4.7.2.10 testTruth()

```
void testTruth (
            bool statement,
            std::string file,
            int line )
```

Tests if the given statement is true.

Parameters

statement	The statement whose truth is to be tested ("1 == 0", for example).
file	The file in which the test is applied (you should be able to just pass in "__FILE__").
line	The line of the file in which the test is applied (you should be able to just pass in "__LINE__").

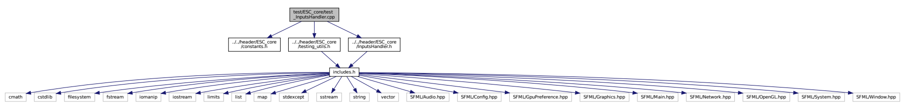
```
{
    if (statement) {
        return;
    }

    std::string error_str = "ERROR: testTruth():\t in ";
    error_str += file;
    error_str += "\tline ";
    error_str += std::to_string(line);
    error_str += ":\t\n";
    error_str += "Given statement is not true";

    #ifdef _WIN32
        std::cout « error_str « std::endl;
    #endif

    throw std::runtime_error(error_str);
    return;
}   /* testTruth() */
```

4.8 test/ESC_core/test_InputsHandler.cpp File Reference

Suite of tests for the InputsHandler class.

```
#include "../../header/ESC_core/constants.h"
#include "../../header/ESC_core/testing_utils.h"
#include "../../header/ESC_core/InputsHandler.h"
```

Include dependency graph for test_InputsHandler.cpp:



Functions

- int main (int argc, char **argv)

4.8.1 Detailed Description

Suite of tests for the InputsHandler class.

A suite of tests for the InputsHandler class.

4.8.2 Function Documentation

4.8.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

```
{
    #ifdef _WIN32
        activateVirtualTerminal();
    #endif  /* _WIN32 */

    printGold("\tTesting InputsHandler");
    std::cout « std::flush;

    srand(time(NULL));
    int n_dots = 8;


    try {
        InputsHandler inputs_handler;

        testFloatEquals(
            int(sf::Keyboard::KeyCount),
            101,
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            inputs_handler.key_press_vec.size(),
            int(sf::Keyboard::KeyCount),
            __FILE__,
            __LINE__
        );

        testFloatEquals(
            inputs_handler.key_pressed_once_vec.size(),
            int(sf::Keyboard::KeyCount),
            __FILE__,
            __LINE__
        );

        sf::Clock clock;
        sf::Event event;
        sf::RenderWindow window(sf::VideoMode(800, 600), "Testing InputsHandler");

        unsigned long long int frame = 0;
        double time_since_run_s = 0;

        while (window.isOpen()) {
            time_since_run_s = clock.getElapsedTime().asSeconds();

            if (
                time_since_run_s >= (frame + 1) * SECONDS_PER_FRAME
            ) {
                while (window.pollEvent(event))
                {
                    inputs_handler.process(&event);

                    if (event.type == sf::Event::Closed) {
                        window.close();
                    }
                }

                window.clear();
                window.display();

                //inputs_handler.printKeysPressed();
                if (inputs_handler.key_pressed_once_vec[sf::Keyboard::Enter]) {
                    std::cout « "Enter" « std::endl;
                }

                std::cout « frame « " : " « time_since_run_s « "\r" « std::flush;

                inputs_handler.reset();
                frame++;
            }
        }
    }


    catch (...) {
        //...

        printGold(" ");
        for (int i = 0; i < n_dots; i++) {
            printGold(".");
        }
        printGold(" ");
        printRed("FAIL");
        std::cout « std::endl;
        throw;
    }


    //...

    printGold(" ");
    for (int i = 0; i < n_dots; i++) {
        printGold(".");
    }
    printGold(" ");
    printGreen("PASS");
    std::cout « std::endl;

    return 0;
}   /* main() */
```

Bibliography

L. Gomila. SFML: Simple and Fast Multimedia Library, 2023. URL `https://www.sfml-dev.org/`. 15

D. van Heesch. Doxygen: Generate documentation from source code, 2023. URL `https://www.doxygen.nl`. 14

Index