# Chess

Generated by Doxygen 1.9.8

# Chapter 1

# Hierarchical Index

## 1.1 Class Hierarchy

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

# 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

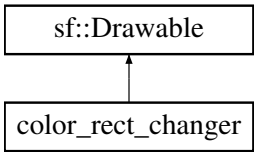
Here is a list of all documented files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 color_rect_changer Class Reference

Inheritance diagram for color_rect_changer:




#### Public Member Functions

- **color_rect_changer** (sf::Vector2f _position, sf::Vector2f _size, std::vector< sf::Color > _color_array, sf::↩ Color _opposite_color)
- void **next** ()
- void **prev** ()
- bool **is_mouse_on** (sf::RenderWindow &window)

#### Public Attributes

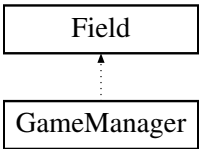
- int **current_ind**
- sf::Vector2f **position**
- sf::Vector2f **size**
- sf::Color **current_value**
- std::vector< sf::Color > **color_array**
- sf::Color **opposite_color**
- sf::RectangleShape **rect**

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

- color_rect_changer.h

## 4.2 Field Struct Reference

Inheritance diagram for Field:




### Public Member Functions

- bool **is_king_nearby** (figurecolor which_one, std::pair< int, int > position)
  *Checks for the presence of a king of a given color in adjacent cells.*
- void **make_turns** (Figure figure, figurecolor current_turn_color)
  *Create a vector of all possible moves of the selected figure.*
- void **simple_make_turns** (Figure figure, figurecolor current_turn_color)
  *Create a vector of all possible moves of the selected figure withou checking on check.*
- void **reset_chsn_and_turns** ()
  *reset turns and choosen*
- Figure **get_figure** (std::pair< int, int > position)
  *retrieves figures from a given position*
- Figure **get_selected** ()
  *retrieves choosen figure*
- bool **is_in_field** (std::pair< int, int > coord)
  *checks whether the figure is on the field*
- int **get_selected_index** ()
  *retrieves index of choosen figure*
- int **get_figure_index** (std::pair< int, int > position)
  *retrieves index of figures from a given position*
- bool **delete_at** (std::pair< int, int > position)
  *deletes a shape at a given position*
- bool **is_check** (figurecolor to_whom)
  *checks on the check*
- bool **is_mate** (figurecolor to_whom)
  *checks for a checkmate*
- Figure **get_king** (figurecolor which_one)
  *getting the king of a given color*
- int **get_king_index** (figurecolor which_one)
  *getting the index of king of a given color*
- bool **check_check** (figurecolor to_whom, std::pair< int, int > from_pos, std::pair< int, int > to_pos)
  *checks on the check in given position*

### Public Attributes

- std::vector< std::pair< int, int > > **turns**
  *Vector of all possible moves of the selected figure.*
- std::vector< Figure > **figures**
  *vector of all figures*

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

- Field.h
- Field.cpp

## 4.3 Figure Class Reference

**Public Member Functions**

- **Figure** (figurenames _name, figurecolor _color)
- figurenames **get_name** ()
  *get name*
- void **set_texture_color** (sf::Color color)
  *set color of figure's texture*
- figurecolor **get_color** ()
  *get color of figure's texture*
- std::vector< std::pair< int, int > > **get_step_conf** ()
  *get configuration of one step*
- std::vector< std::pair< int, int > > **get_take_step_conf** ()
  *get configuration of take in one step*
- std::vector< std::pair< int, int > > **get_line_step_conf** ()
  *get configuration on lines steps*
- std::vector< std::pair< int, int > > **get_take_line_step_conf** ()
  *get configuration on take in lines steps*
- std::pair< int, int > **get_position** ()
  *get position*
- void **set_position** (std::pair< int, int > _position)
  *set position*
- void **draw** (sf::RenderWindow &window)
  *draw*
- bool **is_mouse_over** (sf::RenderWindow &window)
  *check is mouse over figure*

**Public Attributes**

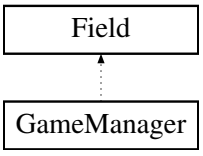
- bool **is_choosen**
  *is figure choosen*
- int **own_cell_size**
  *cell size of figure*

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

- figure.h
- figure.cpp

## 4.4 GameManager Class Reference

Inheritance diagram for GameManager:



**Public Member Functions**

- bool is_window_open ()
- void **Step** ()

  *one frame of game*

### 4.4.1 Member Function Documentation

#### 4.4.1.1 is_window_open()

```
bool GameManager::is_window_open ( )
```

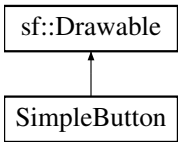
\check on windows is open

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

- GameManager.h
- GameManager.cpp

## 4.5 SimpleButton Class Reference

Inheritance diagram for SimpleButton:




**Public Member Functions**

- **SimpleButton** (sf::Vector2f _position, sf::Vector2f _size, sf::Color _color)
- bool **is_mouse_on** (sf::RenderWindow &window)

**Public Attributes**

- int **current_ind**
- sf::Vector2f **position**
- sf::Vector2f **size**
- sf::RectangleShape **rect**

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

- SimpleButton.h

# Chapter 5

# File Documentation

## 5.1 color_rect_changer.h

```
00001 #include <SFML/Graphics.hpp>
00002 #pragma once
00003 class color_rect_changer : public sf::Drawable
00004 {
00005 public:
00006     int current_ind;
00007     sf::Vector2f position;
00008     sf::Vector2f size;
00009     sf::Color current_value;
00010     std::vector<sf::Color> color_array;
00011     sf::Color opposite_color;
00012     sf::RectangleShape rect;
00013
00014     color_rect_changer(sf::Vector2f _position, sf::Vector2f _size, std::vector<sf::Color>
      _color_array, sf::Color _opposite_color) : color_array(_color_array) {
00015         position = _position;
00016         size = _size;
00017         opposite_color = _opposite_color;
00018         current_ind = 0;
00019         rect.setPosition(position);
00020         rect.setSize(size);
00021         next();
00022         prev();
00023         rect.setFillColor(current_value);
00024     }
00025
00026     void next() {
00027         while (true) {
00028             current_ind++;
00029             if (current_ind >= color_array.size()) {
00030                 current_ind = 0;
00031             }
00032             current_value = color_array[current_ind];
00033             if (current_value != opposite_color)break;
00034         }
00035         rect.setFillColor(current_value);
00036     }
00037
00038     void prev() {
00039         while (true) {
00040             current_ind--;
00041             if (current_ind < 0) {
00042                 current_ind = color_array.size()-1;
00043             }
00044             current_value = color_array[current_ind];
00045             if (current_value != opposite_color)break;
00046         }
00047         rect.setFillColor(current_value);
00048     }
00049
00050     bool is_mouse_on(sf::RenderWindow &window) {
00051         int mouse_x = sf::Mouse::getPosition(window).x;
00052         int mouse_y = sf::Mouse::getPosition(window).y;
00053         int start_pos_x = position.x;
00054         int start_pos_y = position.y;
00055         int end_pos_x = position.x + size.x;
00056         int end_pos_y = position.y + size.y;
```

```
00057         if (mouse_x > start_pos_x && mouse_x < end_pos_x && mouse_y > start_pos_y && mouse_y <
      end_pos_y) {
00058             return true;
00059         }
00060         return false;
00061     }
00062
00063 private:
00064
00065     virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
00066     {
00067         // You can draw other high-level objects
00068         target.draw(rect , states);
00069
00070     }
00071 };
00072
```

## 5.2 Field.h

```
00001 #include "resources.h"
00002 #include "figure.h"
00003 #include <SFML/Graphics.hpp>
00004 #pragma once
00005 struct Field
00006 {
00010     bool is_king_nearby(figurecolor which_one, std::pair<int, int> position);
00014     std::vector<std::pair<int, int» turns;
00018     void make_turns(Figure figure, figurecolor current_turn_color);
00022     void simple_make_turns(Figure figure, figurecolor current_turn_color);
00026     void reset_chsn_and_turns();
00030     std::vector<Figure> figures;
00034     Figure get_figure(std::pair<int, int> position);
00038     Figure get_selected();
00042     bool is_in_field(std::pair<int, int> coord);
00046     int get_selected_index();
00050     int get_figure_index(std::pair<int, int> position);
00054     bool delete_at(std::pair<int, int> position);
00058     bool is_check(figurecolor to_whom);
00062     bool is_mate(figurecolor to_whom);
00066     Figure get_king(figurecolor which_one);
00070     int get_king_index(figurecolor which_one);
00074     bool check_check(figurecolor to_whom, std::pair<int, int> from_pos, std::pair<int, int> to_pos);
00075 };
00076
```

## 5.3 figure.h

```
00001 #include <vector>
00002 #include <SFML/Graphics.hpp>
00003 #include "resources.h"
00004 #pragma once
00005 class Figure
00006 {
00007 private:
00008     std::pair<int, int> position;
00009     sf::Texture texture;
00010     sf::Color texture_color;
00011     sf::Sprite sprite;
00012     figurenames name;
00013     figurecolor color;
00014     std::vector<std::pair<int, int» step_conf;
00015     std::vector<std::pair<int, int» take_step_conf;
00016     std::vector<std::pair<int, int» line_step_conf;
00017     std::vector<std::pair<int, int» take_line_step_conf;
00018     void set_texture();
00019     void pawn_conf();
00020 public:
00024     bool is_choosen;
00028     int own_cell_size;
00029     Figure(figurenames _name,
00030         figurecolor _color);
00034     figurenames get_name();
00038     void set_texture_color(sf::Color color);
00042     figurecolor get_color();
00046     std::vector<std::pair<int, int» get_step_conf();
00050     std::vector<std::pair<int, int» get_take_step_conf();
00054     std::vector<std::pair<int, int» get_line_step_conf();
00058     std::vector<std::pair<int, int» get_take_line_step_conf();
```

```
00062     std::pair<int, int> get_position();
00066     void set_position(std::pair<int, int> _position);
00070     void draw(sf::RenderWindow& window);
00074     bool is_mouse_over(sf::RenderWindow& window);
00075 };
00076
```

## 5.4 GameManager.h

```
00001 #include "resources.h"
00002 #include "figure.h"
00003 #include "Field.h"
00004 #include "color_rect_changer.h"
00005 #include "SimpleButton.h"
00006 #include <ctime>
00007 #include <SFML/Graphics.hpp>
00008 #include <sstream>
00009 #include <stdio.h>
00010 #pragma once
00011 class GameManager : Field
00012 {
00013 private:
00014     int current_cell_size;
00015     bool is_timer;
00016     time_t start_time;
00017     time_t black_time;
00018     time_t white_time;
00019     time_t prev_frame;
00020     time_t cur_frame;
00021     time_t target_time;
00022     time_t decrease_time;
00023     bool is_game_ends;
00024     std::vector<Figure> reserved_turn;
00025     figurecolor reserved_turn_color;
00026     figurecolor who_win;
00027     bool is_playing;
00028     figurecolor current_turn_color;
00029     sf::RenderWindow window;
00030     void win(figurecolor color);
00031     void draw_field();
00032     void turn_turn_back();
00033     void turn_to(std::pair<int, int> position);
00034     bool is_have_check(std::vector<Figure> _figures, figurecolor color);
00035     void init_field();
00036     void draw_figures();
00037     void draw_turns();
00038     void change_turn();
00039     void draw_tile(std::pair<int, int>, sf::Color);
00040     void mouse_clicked();
00041
00042     bool on_mouse_in_tile(std::pair<int, int> position);
00043     void set_figures_texture_color(figurecolor fig_color, sf::Color color);
00044     void set_figures_cell_size(int size);
00045
00046     void reset() {
00047         is_timer = false;
00048         black_time = 0;
00049         white_time = 0;
00050         target_time = 0;
00051         decrease_time = 0;
00052         cur_frame = time(NULL);
00053         start_time = time(NULL);
00054         figures.clear();
00055         who_win = color_none;
00056         is_game_ends = false;
00057         current_turn_color = black;
00058         reserved_turn_color = current_turn_color;
00059         init_field();
00060         set_figures_cell_size(current_cell_size);
00061     }
00062 public:
00063     GameManager() {
00064         current_cell_size = 50;
00065         reset();
00066         window.create(sf::VideoMode(windowsize.first, windowsize.second), window_name,
      sf::Style::Default);
00067     }
00068
00072     bool is_window_open();
00076     void Step();
00077 };
00078
```

## 5.5 resources.h

```
00001 #pragma once
00002 #include <map>
00003 #include <SFML/Graphics.hpp>
00004
00005 #ifndef CONSTANTS_H
00006 #define CONSTANTS_H
00007
00008 static enum figurenames { none, pawn, king, knight, queen, bishop, rook };
00009 static enum figurecolor { black, white, color_none };
00010
00011 static const std::pair<int, int> windowsize = {1000, 600};
00012 static int cell_size = 50;
00013 static const std::pair<int, int> field_start_pos = { 25, 100 };
00014
00015 static sf::Color check_color = sf::Color::Red;
00016 static sf::Color field_cell_color1 = sf::Color(200, 200, 200);
00017 static sf::Color field_cell_color2 = sf::Color(75, 75, 75);
00018 static const std::vector<sf::Color> field_cell_colors = { sf::Color(200, 200, 200), sf::Color(75, 75,
      75), sf::Color(90, 0, 0), sf::Color(0, 90, 0), sf::Color(0, 0, 90)};
00019 static sf::Color field_choosen_color = sf::Color(255, 242, 0, 100);
00020 static sf::Color field_turn_color = sf::Color(50, 242, 50, 100);
00021 static sf::Color field_background_color = sf::Color(150, 150, 150);
00022
00023 static std::string window_name = "Chess";
00024
00025 static std::string textures_path = "textures\\";
00026 static std::map<figurenames, std::string> figure_textures_paths ={
00027     {pawn, textures_path+"pawn.png"},
00028     {king, textures_path + "king.png"},
00029     {knight, textures_path + "knight.png"},
00030     {queen, textures_path + "queen.png"},
00031     {bishop, textures_path + "bishop.png"},
00032     {rook, textures_path + "rook.png"}
00033 };
00034
00035 static std::map<figurecolor, sf::Color> figure_color_and_texture_color_matcher = {
00036     {black, sf::Color::White},
00037     {white, sf::Color::Black}
00038 };
00039 static const std::vector<sf::Color> figure_colors = { sf::Color::Black, sf::Color::Green,
      sf::Color::Yellow, sf::Color::Cyan, sf::Color::White };
00040
00041 static std::vector<figurenames> field_preset = { rook, knight, bishop, queen, king, bishop, knight,
      rook };
00042
00043 static figurecolor reverse_color(figurecolor color) {
00044     if (color == black) {
00045         return white;
00046     }
00047     return black;
00048 }
00049
00050 static std::map<figurenames, std::vector<std::pair<int, int>> step_configurations = {
00051     {knight, {{2,1},{1,2},{-1,2},{2,-1},{1, -2},{-2, 1},{-1, -2}, {-2, -1}}},
00052     {king, {{1,0},{1,1},{0,1},{-1,0},{-1,1},{0,-1},{-1,-1},{1, -1}}}
00053 };
00054
00055 static std::map<figurenames, std::vector<std::pair<int, int>> line_step_configurations = {
00056     {rook, {{1,0},{0,1},{-1,0},{0,-1}}},
00057     {queen, {{1,0},{1,1},{0,1},{-1,0},{-1,1},{0,-1},{-1,-1},{1, -1}}},
00058     {bishop, {{1,1},{-1,1}, {1,-1}, {-1, -1}}}
00059 };
00060
00061 static std::pair<int, int> operator+(std::pair<int, int> a, std::pair<int, int> b) {
00062     return { a.first + b.first, a.second + b.second };
00063 }
00064
00065 static std::pair<int, int> operator-(std::pair<int, int> a, std::pair<int, int> b) {
00066     return { a.first - b.first, a.second - b.second };
00067 }
00068
00069 static bool operator==(std::pair<int, int> a, std::pair<int, int> b) {
00070     return (a.first == b.first && a.second == b.second);
00071 }
00072
00073 #endif
```

## 5.6 SimpleButton.h

```
00001 #include <SFML/Graphics.hpp>
00002 #pragma once
```

```
00003 class SimpleButton : public sf::Drawable
00004 {
00005 public:
00006     int current_ind;
00007     sf::Vector2f position;
00008     sf::Vector2f size;
00009     sf::RectangleShape rect;
00010 
00011     SimpleButton(sf::Vector2f _position, sf::Vector2f _size, sf::Color _color) {
00012         position = _position;
00013         size = _size;
00014         rect.setPosition(position);
00015         rect.setSize(size);
00016         rect.setFillColor(_color);
00017     }
00018 
00019     bool is_mouse_on(sf::RenderWindow& window) {
00020         int mouse_x = sf::Mouse::getPosition(window).x;
00021         int mouse_y = sf::Mouse::getPosition(window).y;
00022         int start_pos_x = position.x;
00023         int start_pos_y = position.y;
00024         int end_pos_x = position.x + size.x;
00025         int end_pos_y = position.y + size.y;
00026         if (mouse_x > start_pos_x && mouse_x < end_pos_x && mouse_y > start_pos_y && mouse_y <
      end_pos_y) {
00027             return true;
00028         }
00029         return false;
00030     }
00031 
00032 private:
00033 
00034     virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const
00035     {
00036         target.draw(rect, states);
00037     }
00038 };
00039 
```

# Index