

MINILASKA 51

1.0.2

Generated by Doxygen 1.9.0



<b>1 Data Structure Index</b>	<b>1</b>
1.1 Data Structures	1
<b>2 File Index</b>	<b>3</b>
2.1 File List	3
<b>3 Data Structure Documentation</b>	<b>5</b>
3.1 board Struct Reference	5
3.1.1 Detailed Description	5
3.1.2 Field Documentation	5
3.1.2.1 mat	5
3.1.2.2 n_cols	6
3.1.2.3 n_rows	6
3.2 pawn Struct Reference	6
3.2.1 Detailed Description	6
3.2.2 Field Documentation	7
3.2.2.1 canMove	7
3.2.2.2 cima	7
3.2.2.3 coordinate	7
3.2.2.4 dim_label	7
3.2.2.5 grade	7
3.2.2.6 isPromoted	8
3.2.2.7 label	8
3.3 player Struct Reference	8
3.3.1 Detailed Description	8
3.3.2 Field Documentation	9
3.3.2.1 color	9
3.3.2.2 dim_pawns	9
3.3.2.3 pawns	9
3.4 point Struct Reference	9
3.4.1 Detailed Description	9
3.4.2 Field Documentation	10
3.4.2.1 x	10
3.4.2.2 y	10
3.5 valueMinimax Struct Reference	10
3.5.1 Detailed Description	10
3.5.2 Field Documentation	11
3.5.2.1 directions	11
3.5.2.2 value	11
<b>4 File Documentation</b>	<b>13</b>
4.1 main.c File Reference	13
4.1.1 Function Documentation	13

4.1.1.1 main()	13
4.2 src/colors/colors.c File Reference	13
4.2.1 Function Documentation	14
4.2.1.1 printColor()	14
4.2.1.2 printTextColor()	14
4.2.1.3 setWhite()	14
4.3 src/colors/colors.h File Reference	15
4.3.1 Detailed Description	16
4.3.2 Macro Definition Documentation	16
4.3.2.1 BBLK	16
4.3.2.2 BBLU	17
4.3.2.3 BCYN	17
4.3.2.4 BGRN	17
4.3.2.5 BHBLK	17
4.3.2.6 BHBLU	17
4.3.2.7 BHCYN	17
4.3.2.8 BHGRN	18
4.3.2.9 BHMAG	18
4.3.2.10 BHRED	18
4.3.2.11 BHWHT	18
4.3.2.12 BHYEL	18
4.3.2.13 BLK	18
4.3.2.14 BLKB	19
4.3.2.15 BLKHB	19
4.3.2.16 BLU	19
4.3.2.17 BLUB	19
4.3.2.18 BLUHB	19
4.3.2.19 BMAG	19
4.3.2.20 BRED	20
4.3.2.21 BWHT	20
4.3.2.22 BYEL	20
4.3.2.23 CYN	20
4.3.2.24 CYNB	20
4.3.2.25 CYNHB	20
4.3.2.26 GRN	21
4.3.2.27 GRNB	21
4.3.2.28 GRNHB	21
4.3.2.29 HBLK	21
4.3.2.30 HBLU	21
4.3.2.31 HCYN	21
4.3.2.32 HGRN	22
4.3.2.33 HMAG	22

4.3.2.34 HRED	22
4.3.2.35 HWHT	22
4.3.2.36 HYEL	22
4.3.2.37 MAG	22
4.3.2.38 MAGB	23
4.3.2.39 MAGHB	23
4.3.2.40 RED	23
4.3.2.41 REDB	23
4.3.2.42 REDHB	23
4.3.2.43 reset	23
4.3.2.44 UBLK	24
4.3.2.45 UBLU	24
4.3.2.46 UCYN	24
4.3.2.47 UGRN	24
4.3.2.48 UMAG	24
4.3.2.49 URED	24
4.3.2.50 UWHT	25
4.3.2.51 UYEL	25
4.3.2.52 WHT	25
4.3.2.53 WHTB	25
4.3.2.54 WHTHB	25
4.3.2.55 YEL	25
4.3.2.56 YELB	26
4.3.2.57 YELHB	26
4.3.3 Function Documentation	26
4.3.3.1 printColor()	26
4.3.3.2 printTextColor()	26
4.3.3.3 setWhite()	27
4.4 src/game_engine/game_engine.c File Reference	27
4.4.1 Function Documentation	28
4.4.1.1 all_blocked()	28
4.4.1.2 can_eat()	29
4.4.1.3 char_converter()	29
4.4.1.4 check_canMove()	30
4.4.1.5 check_char_color()	30
4.4.1.6 check_directions()	31
4.4.1.7 check_player()	31
4.4.1.8 check_spot()	32
4.4.1.9 check_string()	32
4.4.1.10 check_while()	32
4.4.1.11 initialize_board()	33
4.4.1.12 int_converter()	33

4.4.1.13 is_empty()	34
4.4.1.14 is_in()	34
4.4.1.15 is_notstuck()	35
4.4.1.16 is_selected()	35
4.4.1.17 is_victory()	36
4.4.1.18 max_pawns()	36
4.4.1.19 must_eat()	36
4.4.1.20 pawn_promotion()	37
4.4.1.21 print_directions()	37
4.4.1.22 remove_pawn()	38
4.4.1.23 reset_moves_pawns()	38
4.4.1.24 set_moves_pawn()	38
4.4.1.25 update_board()	39
4.4.1.26 uppercase()	39
4.5 src/game_engine/game_engine.h File Reference	39
4.5.1 Detailed Description	41
4.5.2 Typedef Documentation	41
4.5.2.1 coord	42
4.5.2.2 dim_board	42
4.5.2.3 flag	42
4.5.3 Function Documentation	42
4.5.3.1 all_blocked()	42
4.5.3.2 can_eat()	43
4.5.3.3 char_converter()	43
4.5.3.4 check_canMove()	44
4.5.3.5 check_char_color()	44
4.5.3.6 check_directions()	44
4.5.3.7 check_player()	45
4.5.3.8 check_spot()	45
4.5.3.9 check_string()	46
4.5.3.10 check_while()	46
4.5.3.11 initialize_board()	47
4.5.3.12 int_converter()	47
4.5.3.13 is_empty()	48
4.5.3.14 is_in()	48
4.5.3.15 is_notstuck()	48
4.5.3.16 is_selected()	49
4.5.3.17 is_victory()	49
4.5.3.18 max_pawns()	50
4.5.3.19 must_eat()	50
4.5.3.20 pawn_promotion()	51
4.5.3.21 print_directions()	51

4.5.3.22 remove_pawn()	51
4.5.3.23 reset_moves_pawns()	52
4.5.3.24 set_moves_pawn()	52
4.5.3.25 update_board()	53
4.5.3.26 uppercase()	53
4.5.4 Variable Documentation	53
4.5.4.1 board_t	53
4.5.4.2 pawn_t	54
4.5.4.3 player_t	54
4.5.4.4 point_t	54
4.6 src/ia/ia.c File Reference	54
4.6.1 Function Documentation	55
4.6.1.1 call_minimax()	55
4.6.1.2 destroy_value_minimax()	56
4.6.1.3 evaluate_score()	56
4.6.1.4 interrupt_minimax()	56
4.6.1.5 last_move()	58
4.6.1.6 max()	58
4.6.1.7 minimax()	59
4.6.1.8 print_minimax()	59
4.6.1.9 round_ia_minimax()	60
4.6.1.10 round_ia_random()	60
4.7 src/ia/ia.h File Reference	61
4.7.1 Detailed Description	61
4.7.2 Typedef Documentation	62
4.7.2.1 valueMinimax_t	62
4.7.3 Function Documentation	62
4.7.3.1 call_minimax()	62
4.7.3.2 destroy_value_minimax()	63
4.7.3.3 evaluate_score()	63
4.7.3.4 interrupt_minimax()	64
4.7.3.5 last_move()	64
4.7.3.6 max()	65
4.7.3.7 minimax()	65
4.7.3.8 print_minimax()	66
4.7.3.9 round_ia_minimax()	66
4.7.3.10 round_ia_random()	67
4.8 src/memory_management/memory_management.c File Reference	67
4.8.1 Function Documentation	68
4.8.1.1 copy_board()	68
4.8.1.2 create_board()	68
4.8.1.3 create_pawns()	69

4.8.1.4 <code>destroy_board()</code> . . . . .	69
4.8.1.5 <code>destroy_player()</code> . . . . .	69
4.8.1.6 <code>player_copy()</code> . . . . .	70
4.8.1.7 <code>restore_copy()</code> . . . . .	70
4.9 <code>src/memory_management/memory_management.h</code> File Reference . . . . .	71
4.9.1 Detailed Description . . . . .	71
4.9.2 Function Documentation . . . . .	71
4.9.2.1 <code>copy_board()</code> . . . . .	71
4.9.2.2 <code>create_board()</code> . . . . .	72
4.9.2.3 <code>create_pawns()</code> . . . . .	72
4.9.2.4 <code>destroy_board()</code> . . . . .	73
4.9.2.5 <code>destroy_player()</code> . . . . .	73
4.9.2.6 <code>player_copy()</code> . . . . .	74
4.9.2.7 <code>restore_copy()</code> . . . . .	74
4.10 <code>src/movement/movement.c</code> File Reference . . . . .	74
4.10.1 Function Documentation . . . . .	75
4.10.1.1 <code>eat()</code> . . . . .	75
4.10.1.2 <code>move_noeat()</code> . . . . .	76
4.10.1.3 <code>move_p1()</code> . . . . .	77
4.10.1.4 <code>move_p2()</code> . . . . .	78
4.11 <code>src/movement/movement.h</code> File Reference . . . . .	79
4.11.1 Detailed Description . . . . .	79
4.11.2 Function Documentation . . . . .	79
4.11.2.1 <code>eat()</code> . . . . .	80
4.11.2.2 <code>move_noeat()</code> . . . . .	81
4.11.2.3 <code>move_p1()</code> . . . . .	81
4.11.2.4 <code>move_p2()</code> . . . . .	82
4.12 <code>src/user_interaction/user_interaction.c</code> File Reference . . . . .	83
4.12.1 Function Documentation . . . . .	83
4.12.1.1 <code>checkInt()</code> . . . . .	83
4.12.1.2 <code>game()</code> . . . . .	84
4.12.1.3 <code>menu()</code> . . . . .	84
4.12.1.4 <code>print_board()</code> . . . . .	84
4.12.1.5 <code>print_player()</code> . . . . .	85
4.12.1.6 <code>round_choice()</code> . . . . .	85
4.12.1.7 <code>round_player()</code> . . . . .	85
4.12.1.8 <code>while_select_nPawn()</code> . . . . .	86
4.13 <code>src/user_interaction/user_interaction.h</code> File Reference . . . . .	86
4.13.1 Detailed Description . . . . .	87
4.13.2 Function Documentation . . . . .	87
4.13.2.1 <code>checkInt()</code> . . . . .	87
4.13.2.2 <code>game()</code> . . . . .	88



---

4.13.2.3 menu()	88
4.13.2.4 print_board()	88
4.13.2.5 print_player()	89
4.13.2.6 round_choice()	89
4.13.2.7 round_player()	89
4.13.2.8 while_select_nPawn()	90
<b>Index</b>	<b>91</b>



# Chapter 1

## Data Structure Index

### 1.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">board</a>	Description of struct board: . . . . .	5
<a href="#">pawn</a>	Description of struct pawn : . . . . .	6
<a href="#">player</a>	Description of struct player : . . . . .	8
<a href="#">point</a>	Description of struct point : . . . . .	9
<a href="#">valueMinimax</a>	. . . . .	10



## Chapter 2

# File Index

### 2.1 File List

Here is a list of all files with brief descriptions:

<a href="#">main.c</a> . . . . .	13
src/colors/ <a href="#">colors.c</a> . . . . .	13
src/colors/ <a href="#">colors.h</a> . . . . .	15
src/game_engine/ <a href="#">game_engine.c</a> . . . . .	27
src/game_engine/ <a href="#">game_engine.h</a> . . . . .	39
src/ia/ <a href="#">ia.c</a> . . . . .	54
src/ia/ <a href="#">ia.h</a> . . . . .	61
src/memory_management/ <a href="#">memory_management.c</a> . . . . .	67
src/memory_management/ <a href="#">memory_management.h</a> . . . . .	71
src/movement/ <a href="#">movement.c</a> . . . . .	74
src/movement/ <a href="#">movement.h</a> . . . . .	79
src/user_interaction/ <a href="#">user_interaction.c</a> . . . . .	83
src/user_interaction/ <a href="#">user_interaction.h</a> . . . . .	86



## Chapter 3

# Data Structure Documentation

### 3.1 board Struct Reference

Description of struct board:

```
#include <game_engine.h>
```

#### Data Fields

- `char ** mat`
- `dim_board n_rows`
- `dim_board n_cols`

#### 3.1.1 Detailed Description

Description of struct board:

##### Parameters

<code>**mat</code>	: two-dimensional array of characters.
<code>n_rows</code>	: it is the number of row int the matrix.
<code>n_cols</code>	: it is the number of columns int the matrix.

Definition at line 23 of file game\_engine.h.

#### 3.1.2 Field Documentation

##### 3.1.2.1 mat

```
char** mat
```

Definition at line 24 of file game\_engine.h.

### 3.1.2.2 n\_cols

```
dim_board n_cols
```

Definition at line 26 of file game\_engine.h.

### 3.1.2.3 n\_rows

```
dim_board n_rows
```

Definition at line 25 of file game\_engine.h.

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

- src/game\_engine/game\_engine.h

## 3.2 pawn Struct Reference

Description of struct pawn :

```
#include <game_engine.h>
```

### Data Fields

- char \* [label](#)
- unsigned int [dim\\_label](#)
- [point\\_t](#) coordinate
- unsigned int [cima](#)
- unsigned int [grade](#)
- [flag](#) [isPromoted](#)
- unsigned int \* [canMove](#)

### 3.2.1 Detailed Description

Description of struct pawn :

#### Parameters

<i>*label</i>	: it is a char pointer,so it is the label of a pawn in the chessboard.
<i>dim_label</i>	: it is the number of char that are necessary to represent the number of a pawn in the label.
<i>coordinate</i>	: they are the coordinates of a pawn in the board.
<i>cima</i>	: it is the position of the highest pawn int he label. <span style="float: right;">Generated by Doxygen</span>
<i>grade</i>	: if 0 then the label doesn't appear in the board,else it represents the height of the pawn tower.
<i>is_promoted</i>	: if 1 then the pawn can move to bottom right or bottom left respect the pawn ,else the pawn can move only left or right respect the pawn.



Definition at line 56 of file game\_engine.h.

### 3.2.2 Field Documentation

#### 3.2.2.1 canMove

```
unsigned int* canMove
```

Definition at line 63 of file game\_engine.h.

#### 3.2.2.2 cima

```
unsigned int cima
```

Definition at line 60 of file game\_engine.h.

#### 3.2.2.3 coordinate

```
point_t coordinate
```

Definition at line 59 of file game\_engine.h.

#### 3.2.2.4 dim\_label

```
unsigned int dim_label
```

Definition at line 58 of file game\_engine.h.

#### 3.2.2.5 grade

```
unsigned int grade
```

Definition at line 61 of file game\_engine.h.

### 3.2.2.6 isPromoted

`flag isPromoted`

Definition at line 62 of file `game_engine.h`.

### 3.2.2.7 label

`char* label`

label of the pawn ex. | BN07 |

Definition at line 57 of file `game_engine.h`.

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

- `src/game_engine/game_engine.h`

## 3.3 player Struct Reference

Description of struct player :

```
#include <game_engine.h>
```

### Data Fields

- char `color`
- `pawn_t` \* `pawns`
- unsigned int `dim_pawns`

### 3.3.1 Detailed Description

Description of struct player :

#### Parameters

<i>color</i>	: it represents the character of the player,eventually the color when the board is printed.
<i>*pawns</i>	: it is a pointer of struct pawn,so it is the pawn array.
<i>dim_pawns</i>	: it is the number of pawns of the player,so it is the dimension of pawns array.

Definition at line 74 of file `game_engine.h`.

### 3.3.2 Field Documentation

#### 3.3.2.1 color

```
char color
```

Definition at line 75 of file game\_engine.h.

#### 3.3.2.2 dim\_pawns

```
unsigned int dim_pawns
```

Definition at line 77 of file game\_engine.h.

#### 3.3.2.3 pawns

```
pawn\_t* pawns
```

Definition at line 76 of file game\_engine.h.

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

- [src/game\\_engine/game\\_engine.h](#)

## 3.4 point Struct Reference

Description of struct point :

```
#include <game_engine.h>
```

### Data Fields

- [coord x](#)
- [coord y](#)

#### 3.4.1 Detailed Description

Description of struct point :

**Parameters**

<code>x</code>	: it represents the coordinate x of the matrix,so the column coordinate.
<code>y</code>	: it represents the coordinate y of the matrix,so the row coordinate.

Definition at line 36 of file game\_engine.h.

### 3.4.2 Field Documentation

#### 3.4.2.1 `x`

`coord x`

`cols`

Definition at line 37 of file game\_engine.h.

#### 3.4.2.2 `y`

`coord y`

`rows`

Definition at line 38 of file game\_engine.h.

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

- [src/game\\_engine/game\\_engine.h](#)

## 3.5 valueMinimax Struct Reference

### Data Fields

- `int value`
- `char * directions`

#### 3.5.1 Detailed Description

Definition at line 16 of file ia.c.

## 3.5.2 Field Documentation

### 3.5.2.1 directions

```
char* directions
```

Definition at line 18 of file ia.c.

### 3.5.2.2 value

```
int value
```

Definition at line 17 of file ia.c.

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

- [src/ia/ia.c](#)



## Chapter 4

# File Documentation

### 4.1 main.c File Reference

```
#include <stdio.h>
#include "src/game_engine/game_engine.h"
#include "src/user_interaction/user_interaction.h"
```

#### Functions

- int [main](#) ()

#### 4.1.1 Function Documentation

##### 4.1.1.1 main()

```
int main ( )
```

Definition at line 8 of file main.c.

### 4.2 src/colors/colors.c File Reference

```
#include <stdio.h>
#include "colors.h"
```

## Functions

- void `printTextColor` (char color)  
*Set the next text based on the player color.*
- void `printColor` (char color)  
*Set the next background color based on the player color.*
- void `setWhite` ()  
*Set the next text to white.*

### 4.2.1 Function Documentation

#### 4.2.1.1 `printColor()`

```
void printColor (  
    char color )
```

Set the next background color based on the player color.

##### Parameters

<i>color</i>	Color of the player.
--------------	----------------------

Definition at line 26 of file colors.c.

#### 4.2.1.2 `printTextColor()`

```
void printTextColor (  
    char color )
```

Set the next text based on the player color.

##### Parameters

<i>color</i>	Color of the player.
--------------	----------------------

Definition at line 5 of file colors.c.

#### 4.2.1.3 `setWhite()`

```
void setWhite ( )
```

Set the next text to white.

Definition at line 47 of file colors.c.



## 4.3 src/colors/colors.h File Reference

### Macros

- #define **BLK** "\033[0;30m"
- #define **RED** "\033[0;31m"
- #define **GRN** "\033[0;32m"
- #define **YEL** "\033[0;33m"
- #define **BLU** "\033[0;34m"
- #define **MAG** "\033[0;35m"
- #define **CYN** "\033[0;36m"
- #define **WHT** "\033[0;37m"
- #define **BBLK** "\033[1;30m"
- #define **BRED** "\033[1;31m"
- #define **BGRN** "\033[1;32m"
- #define **BYEL** "\033[1;33m"
- #define **BBLU** "\033[1;34m"
- #define **BMAG** "\033[1;35m"
- #define **BCYN** "\033[1;36m"
- #define **BWHT** "\033[1;37m"
- #define **UBLK** "\033[4;30m"
- #define **URED** "\033[4;31m"
- #define **UGRN** "\033[4;32m"
- #define **UYEL** "\033[4;33m"
- #define **UBLU** "\033[4;34m"
- #define **UMAG** "\033[4;35m"
- #define **UCYN** "\033[4;36m"
- #define **UWHT** "\033[4;37m"
- #define **BLKB** "\033[40m"
- #define **REDB** "\033[41m"
- #define **GRNB** "\033[42m"
- #define **YELB** "\033[43m"
- #define **BLUB** "\033[44m"
- #define **MAGB** "\033[45m"
- #define **CYNB** "\033[46m"
- #define **WHTB** "\033[47m"
- #define **BLKHB** "\033[0;100m"
- #define **REDHB** "\033[0;101m"
- #define **GRNHB** "\033[0;102m"
- #define **YELHB** "\033[0;103m"
- #define **BLUHB** "\033[0;104m"
- #define **MAGHB** "\033[0;105m"
- #define **CYNHB** "\033[0;106m"
- #define **WHTHB** "\033[0;107m"
- #define **HBLK** "\033[0;90m"
- #define **HRED** "\033[0;91m"
- #define **HGRN** "\033[0;92m"
- #define **HYEL** "\033[0;93m"
- #define **HBLU** "\033[0;94m"
- #define **HMAG** "\033[0;95m"
- #define **HCYN** "\033[0;96m"
- #define **HWHT** "\033[0;97m"
- #define **BHBLK** "\033[1;90m"
- #define **BHRED** "\033[1;91m"

- `#define BHGRN "\033[1;92m"`
- `#define BHYEL "\033[1;93m"`
- `#define BHBLU "\033[1;94m"`
- `#define BHMAG "\033[1;95m"`
- `#define BHCYN "\033[1;96m"`
- `#define BHWHT "\033[1;97m"`
- `#define reset "\033[0m"`

## Functions

- void `setWhite` ()  
*Set the next text to white.*
- void `printTextColor` (char color)  
*Set the next text based on the player color.*
- void `printColor` (char color)  
*Set the next background color based on the player color.*

### 4.3.1 Detailed Description

#### Author

Diego Passarella, Davide Pasqual, Michelle Ravagnan

#### Version

1.0.2

#### Date

2021-01-17

#### Copyright

Copyright (c) 2021

### 4.3.2 Macro Definition Documentation

#### 4.3.2.1 BBLK

```
#define BBLK "\033[1;30m"
```

Definition at line 25 of file colors.h.

#### 4.3.2.2 BBLU

```
#define BBLU "\033[1;34m"
```

Definition at line 29 of file colors.h.

#### 4.3.2.3 BCYN

```
#define BCYN "\033[1;36m"
```

Definition at line 31 of file colors.h.

#### 4.3.2.4 BGRN

```
#define BGRN "\033[1;32m"
```

Definition at line 27 of file colors.h.

#### 4.3.2.5 BHBLK

```
#define BHBLK "\033[1;90m"
```

Definition at line 75 of file colors.h.

#### 4.3.2.6 BHBLU

```
#define BHBLU "\033[1;94m"
```

Definition at line 79 of file colors.h.

#### 4.3.2.7 BHCYN

```
#define BHCYN "\033[1;96m"
```

Definition at line 81 of file colors.h.

#### 4.3.2.8 BHGRN

```
#define BHGRN "\033[1;92m"
```

Definition at line 77 of file colors.h.

#### 4.3.2.9 BHMAG

```
#define BHMAG "\033[1;95m"
```

Definition at line 80 of file colors.h.

#### 4.3.2.10 BHRED

```
#define BHRED "\033[1;91m"
```

Definition at line 76 of file colors.h.

#### 4.3.2.11 BHWHT

```
#define BHWHT "\033[1;97m"
```

Definition at line 82 of file colors.h.

#### 4.3.2.12 BHYEL

```
#define BHYEL "\033[1;93m"
```

Definition at line 78 of file colors.h.

#### 4.3.2.13 BLK

```
#define BLK "\033[0;30m"
```

Definition at line 15 of file colors.h.

#### 4.3.2.14 BLKB

```
#define BLKB "\033[40m"
```

Definition at line 45 of file colors.h.

#### 4.3.2.15 BLKHB

```
#define BLKHB "\033[0;100m"
```

Definition at line 55 of file colors.h.

#### 4.3.2.16 BLU

```
#define BLU "\033[0;34m"
```

Definition at line 19 of file colors.h.

#### 4.3.2.17 BLUB

```
#define BLUB "\033[44m"
```

Definition at line 49 of file colors.h.

#### 4.3.2.18 BLUHB

```
#define BLUHB "\033[0;104m"
```

Definition at line 59 of file colors.h.

#### 4.3.2.19 BMAG

```
#define BMAG "\033[1;35m"
```

Definition at line 30 of file colors.h.

#### 4.3.2.20 BRED

```
#define BRED "\033[1;31m"
```

Definition at line 26 of file colors.h.

#### 4.3.2.21 BWHT

```
#define BWHT "\033[1;37m"
```

Definition at line 32 of file colors.h.

#### 4.3.2.22 BYEL

```
#define BYEL "\033[1;33m"
```

Definition at line 28 of file colors.h.

#### 4.3.2.23 CYN

```
#define CYN "\033[0;36m"
```

Definition at line 21 of file colors.h.

#### 4.3.2.24 CYNB

```
#define CYNB "\033[46m"
```

Definition at line 51 of file colors.h.

#### 4.3.2.25 CYNHB

```
#define CYNHB "\033[0;106m"
```

Definition at line 61 of file colors.h.

#### 4.3.2.26 GRN

```
#define GRN "\033[0;32m"
```

Definition at line 17 of file colors.h.

#### 4.3.2.27 GRNB

```
#define GRNB "\033[42m"
```

Definition at line 47 of file colors.h.

#### 4.3.2.28 GRNHB

```
#define GRNHB "\033[0;102m"
```

Definition at line 57 of file colors.h.

#### 4.3.2.29 HBLK

```
#define HBLK "\033[0;90m"
```

Definition at line 65 of file colors.h.

#### 4.3.2.30 HBLU

```
#define HBLU "\033[0;94m"
```

Definition at line 69 of file colors.h.

#### 4.3.2.31 HCYN

```
#define HCYN "\033[0;96m"
```

Definition at line 71 of file colors.h.

**4.3.2.32 HGRN**

```
#define HGRN "\033[0;92m"
```

Definition at line 67 of file colors.h.

**4.3.2.33 HMAG**

```
#define HMAG "\033[0;95m"
```

Definition at line 70 of file colors.h.

**4.3.2.34 HRED**

```
#define HRED "\033[0;91m"
```

Definition at line 66 of file colors.h.

**4.3.2.35 HWHT**

```
#define HWHT "\033[0;97m"
```

Definition at line 72 of file colors.h.

**4.3.2.36 HYEL**

```
#define HYEL "\033[0;93m"
```

Definition at line 68 of file colors.h.

**4.3.2.37 MAG**

```
#define MAG "\033[0;35m"
```

Definition at line 20 of file colors.h.



#### 4.3.2.38 MAGB

```
#define MAGB "\033[45m"
```

Definition at line 50 of file colors.h.

#### 4.3.2.39 MAGHB

```
#define MAGHB "\033[0;105m"
```

Definition at line 60 of file colors.h.

#### 4.3.2.40 RED

```
#define RED "\033[0;31m"
```

Definition at line 16 of file colors.h.

#### 4.3.2.41 REDB

```
#define REDB "\033[41m"
```

Definition at line 46 of file colors.h.

#### 4.3.2.42 REDHB

```
#define REDHB "\033[0;101m"
```

Definition at line 56 of file colors.h.

#### 4.3.2.43 reset

```
#define reset "\033[0m"
```

Definition at line 85 of file colors.h.

**4.3.2.44 UBLK**

```
#define UBLK "\033[4;30m"
```

Definition at line 35 of file colors.h.

**4.3.2.45 UBLU**

```
#define UBLU "\033[4;34m"
```

Definition at line 39 of file colors.h.

**4.3.2.46 UCYN**

```
#define UCYN "\033[4;36m"
```

Definition at line 41 of file colors.h.

**4.3.2.47 UGRN**

```
#define UGRN "\033[4;32m"
```

Definition at line 37 of file colors.h.

**4.3.2.48 UMAG**

```
#define UMAG "\033[4;35m"
```

Definition at line 40 of file colors.h.

**4.3.2.49 URED**

```
#define URED "\033[4;31m"
```

Definition at line 36 of file colors.h.

#### 4.3.2.50 UWHT

```
#define UWHT "\033[4;37m"
```

Definition at line 42 of file colors.h.

#### 4.3.2.51 UYEL

```
#define UYEL "\033[4;33m"
```

Definition at line 38 of file colors.h.

#### 4.3.2.52 WHT

```
#define WHT "\033[0;37m"
```

Definition at line 22 of file colors.h.

#### 4.3.2.53 WHTB

```
#define WHTB "\033[47m"
```

Definition at line 52 of file colors.h.

#### 4.3.2.54 WHTHB

```
#define WHTHB "\033[0;107m"
```

Definition at line 62 of file colors.h.

#### 4.3.2.55 YEL

```
#define YEL "\033[0;33m"
```

Definition at line 18 of file colors.h.

#### 4.3.2.56 YELB

```
#define YELB "\033[43m"
```

Definition at line 48 of file colors.h.

#### 4.3.2.57 YELHB

```
#define YELHB "\033[0;103m"
```

Definition at line 58 of file colors.h.

### 4.3.3 Function Documentation

#### 4.3.3.1 printColor()

```
void printColor (  
    char color )
```

Set the next background color based on the player color.

##### Parameters

<i>color</i>	Color of the player.
--------------	----------------------

Definition at line 26 of file colors.c.

#### 4.3.3.2 printTextColor()

```
void printTextColor (  
    char color )
```

Set the next text based on the player color.

##### Parameters

<i>color</i>	Color of the player.
--------------	----------------------

Definition at line 5 of file colors.c.

#### 4.3.3.3 setWhite()

```
void setWhite ( )
```

Set the next text to white.

Definition at line 47 of file colors.c.

## 4.4 src/game\_engine/game\_engine.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "../colors/colors.h"
#include "game_engine.h"
#include "../ia/ia.h"
#include "../memory_management/memory_management.h"
#include "../movement/movement.h"
#include "../user_interaction/user_interaction.h"
```

## Functions

- void [initialize\\_board](#) ([board\\_t](#) \*board, unsigned int cifre)  
*Set char on the board.*
- void [print\\_directions](#) (unsigned int \*arr, unsigned int dim, unsigned int np)  
*Print hints for the movements of a given pawn.*
- void [update\\_board](#) ([board\\_t](#) \*t, [player\\_t](#) \*player)  
*Update the board after a player move.*
- unsigned int [check\\_spot](#) ([board\\_t](#) board, unsigned int row, unsigned int col, unsigned int dim\_label)  
*Check if in a given coordinate on the board there is a pawn or not.*
- unsigned int [is\\_selected](#) ([player\\_t](#) \*player, int num\_pawn, unsigned int nPI)  
*Check if a given pawn exists in the board.*
- void [remove\\_pawn](#) ([board\\_t](#) \*board, unsigned int row, unsigned int col, unsigned int dim\_label)  
*Reset label from a given coordinate.*
- void [pawn\\_promotion](#) ([player\\_t](#) \*players, unsigned int num\_pawn, unsigned int nPI, unsigned int last\_row)  
*Promote a given pawn.*
- int [check\\_player](#) ([player\\_t](#) \*players, unsigned int x, unsigned int y, unsigned int nPI)  
*Check if in a given coordinate there is an enemy pawn.*
- unsigned int [is\\_in](#) (int r, int c, [board\\_t](#) board)  
*Check if a given coordinate are included in the board.*
- unsigned int [check\\_directions](#) (unsigned int \*canMove\_arr, unsigned int dim\_canMove, char \*str)  
*Check if a given string is setted in the pawn's possible movements.*
- void [must\\_eat](#) ([player\\_t](#) \*players, [board\\_t](#) \*board, unsigned int n\_pawn, unsigned int nPI)  
*Set mandatory eat if possible.*
- int [can\\_eat](#) ([player\\_t](#) \*players, unsigned int num\_pawn, char \*str, [board\\_t](#) \*board, unsigned int nPI)  
*Check enemy pawns in a given direction.*

- unsigned int `all_blocked` (`player_t` \*players, unsigned int nPI)  
*Check if a player can move.*
- unsigned int `is_victory` (`player_t` \*players)  
*Check if there is a winner.*
- int `is_empty` (`player_t` p)  
*Check if in the player's array there is an unused pawn.*
- unsigned int `max_pawns` (unsigned int r, unsigned int c)  
*Calculate the max number of pawns in a given space.*
- unsigned int `is_notstuck` (`player_t` \*players, `board_t` board, unsigned int nPawn, unsigned int nPI)  
*Set possible movements of a given pawn.*
- void `set_moves_pawn` (`player_t` \*players, `board_t` \*board, unsigned int nPI, int nPawn)  
*Set possible movements of a given pawn and set mandatory eat.*
- void `reset_moves_pawns` (`player_t` \*players, int nPawn, unsigned int nPI)  
*Reset possible movements of a given pawn.*
- unsigned int `check_canMove` (`player_t` \*players, unsigned int nPed, unsigned int nPI)  
*Check if a pawn is able to move.*
- unsigned int `check_while` (`player_t` \*players, unsigned int nPI, unsigned int nPawn)  
*Check if a pawn is selectable.*
- unsigned int `check_string` (char \*str)  
*Compare if the a given string equals to a valid movement.*
- unsigned int `char_converter` (`board_t` board, unsigned int r, unsigned int c, unsigned int dim\_label)  
*Convert a given char on a label to int.*
- char `int_converter` (int num, unsigned int index)  
*Convert a given int to a char.*
- char `uppercase` (char c)  
*Uppercase a given char.*
- unsigned int `check_char_color` (char c)  
*Check if a given char matches a color.*

## 4.4.1 Function Documentation

### 4.4.1.1 `all_blocked()`

```
unsigned int all_blocked (
    player_t * players,
    unsigned int nPI )
```

Check if a player can move.

#### Parameters

<i>players</i>	Pointer to the players array.
<i>nPI</i>	Number of the player.

### Returns

unsigned int Return 1 if there is at least one pawn can move, 0 else.

Definition at line 344 of file game\_engine.c.

#### 4.4.1.2 can\_eat()

```
int can_eat (
    player_t * players,
    unsigned int num_pawn,
    char * str,
    board_t * board,
    unsigned int nPl )
```

Check enemy pawns in a given direction.

### Parameters

<i>players</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to check.
<i>str</i>	Direction to check from the player's pawn.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of the player.

### Returns

int Return enemy pawn number that can be eaten, else negative numbers.

Definition at line 222 of file game\_engine.c.

#### 4.4.1.3 char\_converter()

```
unsigned int char_converter (
    board_t board,
    unsigned int r,
    unsigned int c,
    unsigned int dim_label )
```

Convert a given char on a label to int.

### Parameters

<i>board</i>	Instance of the board.
<i>r</i>	Number of the row (y).
<i>c</i>	Number of the column (x).
<i>dim_label</i>	Dimension of the label of the pawn.

**Returns**

unsigned int Return the conversion of the char to int.

Definition at line 568 of file game\_engine.c.

**4.4.1.4 check\_canMove()**

```
unsigned int check_canMove (  
    player_t * players,  
    unsigned int nPed,  
    unsigned int nPl )
```

Check if a pawn is able to move.

**Parameters**

<i>players</i>	Pointer to the players array.
<i>nPed</i>	Number of the pawn to check.
<i>nPl</i>	Number of the player.

**Returns**

unsigned int Return 1 if the pawn can move, 0 otherwise.

Definition at line 528 of file game\_engine.c.

**4.4.1.5 check\_char\_color()**

```
unsigned int check_char_color (  
    char c )
```

Check if a given char matches a color.

**Parameters**

<i>c</i>	Char to check.
----------	----------------

**Returns**

unsigned int Return 1 if the char matches, 2 if the char mathes being uppercase, 0 otherwise.

Definition at line 591 of file game\_engine.c.



#### 4.4.1.6 check\_directions()

```
unsigned int check_directions (
    unsigned int * canMove_arr,
    unsigned int dim_canMove,
    char * str )
```

Check if a given string is setted in the pawn's possible movements.

##### Parameters

<i>canMove_arr</i>	Pointer to the canMove array of a pawn.
<i>dim_canMove</i>	Dimension of the canMove array, 4 if the pawn is promoted, 2 else.
<i>str</i>	String of the direction to compare.

##### Returns

unsigned int Return 1 if the corresponding string is setted on the canMove array, 0 else.

Definition at line 154 of file game\_engine.c.

#### 4.4.1.7 check\_player()

```
int check_player (
    player_t * players,
    unsigned int x,
    unsigned int y,
    unsigned int nPl )
```

Check if in a given coordinate there is an enemy pawn.

##### Parameters

<i>players</i>	Pointer to the players array.
<i>x</i>	Number of the column (x).
<i>y</i>	Number of the row (y).
<i>nPl</i>	Number of the player.

##### Returns

int Return -1 if a there isn't any enemy pawn, else the number of the enemy player.

Definition at line 131 of file game\_engine.c.

#### 4.4.1.8 check\_spot()

```
unsigned int check_spot (
    board_t board,
    unsigned int row,
    unsigned int col,
    unsigned int dim_label )
```

Check if in a given coordinate on the board there is a pawn or not.

##### Parameters

<i>board</i>	Instance of the board.
<i>row</i>	Number of the row (y).
<i>col</i>	Number of the column (x).
<i>dim_label</i>	Total number of space stored for the label of the pawn.

##### Returns

unsigned int Return 1 if there is a pawn, 0 else.

Definition at line 92 of file game\_engine.c.

#### 4.4.1.9 check\_string()

```
unsigned int check_string (
    char * str )
```

Compare if the a given string equals to a valid movement.

##### Parameters

<i>str</i>	String to compare.
------------	--------------------

##### Returns

unsigned int Return 1 if the string correspond, 0 otherwise.

Definition at line 551 of file game\_engine.c.

#### 4.4.1.10 check\_while()

```
unsigned int check_while (
    player_t * players,
    unsigned int nPl,
    unsigned int nPawn )
```

Check if a pawn is selectable.

## Parameters

<i>players</i>	Pointer to the players array.
<i>nPl</i>	Number of the player.
<i>nPawn</i>	Number of the pawn to check.

## Returns

unsigned int Return 1 if the pawn is selectable, 0 otherwise.

Definition at line 543 of file game\_engine.c.

#### 4.4.1.11 initialize\_board()

```
void initialize_board (
    board_t * board,
    unsigned int cifre )
```

Set char on the board.

'#' white cell, it's where pawns can move ' ' black cell, it's where pawns cannot move

## Parameters

<i>board</i>	Pointer to the board.
<i>cifre</i>	Number of columns for the label.

Definition at line 21 of file game\_engine.c.

#### 4.4.1.12 int\_converter()

```
char int_converter (
    int num,
    unsigned int index )
```

Convert a given int to a char.

## Parameters

<i>num</i>	Number to be converted.
<i>index</i>	Index of the pow of 10.

**Returns**

char Return the conversion of the int to a char.

Definition at line 577 of file game\_engine.c.

**4.4.1.13 is\_empty()**

```
int is_empty (
    player_t p )
```

Check if in the player's array there is an unused pawn.

**Parameters**

<i>p</i>	Instance of a player.
----------	-----------------------

**Returns**

int Return the position of the first unused pawn in the array, -1 else.

Definition at line 384 of file game\_engine.c.

**4.4.1.14 is\_in()**

```
unsigned int is_in (
    int r,
    int c,
    board_t board )
```

Check if a given coordinate are included in the board.

**Parameters**

<i>r</i>	Number of the row (y).
<i>c</i>	Number of the column (x).
<i>board</i>	Instance of the board.

**Returns**

unsigned int Return 1 if the coordinate are inside the board, 0 else.

Definition at line 146 of file game\_engine.c.

#### 4.4.1.15 is\_notstuck()

```
unsigned int is_notstuck (
    player_t * players,
    board_t board,
    unsigned int nPawn,
    unsigned int nPl )
```

Set possible movements of a given pawn.

##### Parameters

<i>players</i>	Pointer to the players array.
<i>board</i>	Instance of the board.
<i>nPawn</i>	Number of the pawn to set.
<i>nPl</i>	Number of the player.

##### Returns

unsigned int Return 1 if the function runs succesfully, 0 else.

Definition at line 420 of file game\_engine.c.

#### 4.4.1.16 is\_selected()

```
unsigned int is_selected (
    player_t * player,
    int num_pawn,
    unsigned int nPl )
```

Check if a given pawn exists in the board.

##### Parameters

<i>player</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to check.
<i>nPl</i>	Number of the player.

##### Returns

unsigned int Return 1 if the pawn exists in the board, 0 else.

Definition at line 104 of file game\_engine.c.

#### 4.4.1.17 is\_victory()

```
unsigned int is_victory (
    player_t * players )
```

Check if there is a winner.

##### Parameters

<i>players</i>	Pointer to the players array.
----------------	-------------------------------

##### Returns

unsigned int Return 1 if player1 won, 2 if player2 won, 17 else.

Definition at line 354 of file game\_engine.c.

#### 4.4.1.18 max\_pawns()

```
unsigned int max_pawns (
    unsigned int r,
    unsigned int c )
```

Calculate the max number of pawns in a given space.

##### Parameters

<i>r</i>	Total number of rows.
<i>c</i>	Total number of columns.

##### Returns

unsigned int Return the max number of pawns available.

Definition at line 394 of file game\_engine.c.

#### 4.4.1.19 must\_eat()

```
void must_eat (
    player_t * players,
    board_t * board,
    unsigned int n_pawn,
    unsigned int nPl )
```

Set mandatory eat if possible.

## Parameters

<i>players</i>	Pointer to the players array.
<i>board</i>	Pointer to the board.
<i>n_pawn</i>	Number of the pawn to check.
<i>nPl</i>	Number of the player.

Definition at line 183 of file game\_engine.c.

#### 4.4.1.20 pawn\_promotion()

```
void pawn_promotion (
    player_t * players,
    unsigned int num_pawn,
    unsigned int nPl,
    unsigned int last_row )
```

Promote a given pawn.

## Parameters

<i>players</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to promote.
<i>nPl</i>	Number of the player.
<i>last_row</i>	Check if a player pawn reaches the end of the board from the perspective of the player.

Definition at line 122 of file game\_engine.c.

#### 4.4.1.21 print\_directions()

```
void print_directions (
    unsigned int * arr,
    unsigned int dim,
    unsigned int np )
```

Print hints for the movements of a given pawn.

## Parameters

<i>arr</i>	Array of the available movements of a pawn
<i>dim</i>	Dimension of the array 'arr', 4 if the pawn is promoted, 2 if the pawn isn't promoted.
<i>np</i>	Number of the pawn.

Definition at line 44 of file game\_engine.c.

#### 4.4.1.22 remove\_pawn()

```
void remove_pawn (
    board_t * board,
    unsigned int row,
    unsigned int col,
    unsigned int dim_label )
```

Reset label from a given coordinate.

##### Parameters

<i>board</i>	Pointer to the board.
<i>row</i>	Number of the row (y).
<i>col</i>	Number of the column (x).
<i>dim_label</i>	Total number of space stored for the label of the pawn.

Definition at line 112 of file game\_engine.c.

#### 4.4.1.23 reset\_moves\_pawns()

```
void reset_moves_pawns (
    player_t * players,
    int nPawn,
    unsigned int nPl )
```

Reset possible movements of a given pawn.

##### Parameters

<i>players</i>	Pointer to the players array.
<i>nPawn</i>	If -1 reset movements of all pawns of a player, else reset movements of the given pawn.
<i>nPl</i>	Number of the player.

Definition at line 501 of file game\_engine.c.

#### 4.4.1.24 set\_moves\_pawn()

```
void set_moves_pawn (
    player_t * players,
    board_t * board,
    unsigned int nPl,
    int nPawn )
```

Set possible movements of a given pawn and set mandatory eat.

Calls to is\_notstuck, must\_eat and reset\_moves\_pawns.



## Parameters

<i>players</i>	Pointer to the players array.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of the player.
<i>nPawn</i>	If -1 set movements of all pawns of a player, else set movements of the given pawn.

Definition at line 477 of file game\_engine.c.

## 4.4.1.25 update\_board()

```
void update_board (
    board_t * t,
    player_t * player )
```

Update the board after a player move.

## Parameters

<i>t</i>	Pointer to the board.
<i>player</i>	Pointer of players array.

Definition at line 74 of file game\_engine.c.

## 4.4.1.26 uppercase()

```
char uppercase (
    char c )
```

Uppercase a given char.

## Parameters

<i>c</i>	Char to be uppercased.
----------	------------------------

## Returns

char Return the uppercased char.

Definition at line 587 of file game\_engine.c.

## 4.5 src/game\_engine/game\_engine.h File Reference

## Data Structures

- struct [board](#)

*Description of struct board:*

- struct [point](#)

*Description of struct point :*

- struct [pawn](#)

*Description of struct pawn :*

- struct [player](#)

*Description of struct player :*

## Typedefs

- typedef unsigned int [dim\\_board](#)
- typedef unsigned int [coord](#)
- typedef unsigned int [flag](#)

## Functions

- void [initialize\\_board](#) ([board\\_t](#) \*board, unsigned int cifre)  
*Set char on the board.*
- void [print\\_directions](#) (unsigned int \*arr, unsigned int dim, unsigned int np)  
*Print hints for the movements of a given pawn.*
- void [update\\_board](#) ([board\\_t](#) \*t, [player\\_t](#) \*player)  
*Update the board after a player move.*
- unsigned int [check\\_spot](#) ([board\\_t](#) board, unsigned int row, unsigned int col, unsigned int dim\_label)  
*Check if in a given coordinate on the board there is a pawn or not.*
- unsigned int [is\\_selected](#) ([player\\_t](#) \*player, int num\_pawn, unsigned int nPI)  
*Check if a given pawn exists in the board.*
- void [remove\\_pawn](#) ([board\\_t](#) \*board, unsigned int row, unsigned int col, unsigned int dim\_label)  
*Reset label from a given coordinate.*
- void [pawn\\_promotion](#) ([player\\_t](#) \*players, unsigned int num\_pawn, unsigned int nPI, unsigned int last\_row)  
*Promote a given pawn.*
- int [check\\_player](#) ([player\\_t](#) \*players, unsigned int x, unsigned int y, unsigned int nPI)  
*Check if in a given coordinate there is an enemy pawn.*
- unsigned int [is\\_in](#) (int r, int c, [board\\_t](#) board)  
*Check if a given coordinate are included in the board.*
- unsigned int [check\\_directions](#) (unsigned int \*canMove\_arr, unsigned int dim\_canMove, char \*str)  
*Check if a given string is setted in the pawn's possible movements.*
- void [must\\_eat](#) ([player\\_t](#) \*players, [board\\_t](#) \*board, unsigned int n\_pawn, unsigned int nPI)  
*Set mandatory eat if possible.*
- int [can\\_eat](#) ([player\\_t](#) \*players, unsigned int num\_pawn, char \*str, [board\\_t](#) \*board, unsigned int nPI)  
*Check enemy pawns in a given direction.*
- unsigned int [all\\_blocked](#) ([player\\_t](#) \*players, unsigned int nPI)  
*Check if a player can move.*
- unsigned int [is\\_victory](#) ([player\\_t](#) \*players)  
*Check if there is a winner.*
- int [is\\_empty](#) ([player\\_t](#) p)  
*Check if in the player's array there is an unused pawn.*
- unsigned int [max\\_pawns](#) (unsigned int r, unsigned int c)  
*Calculate the max number of pawns in a given space.*
- unsigned int [is\\_notstuck](#) ([player\\_t](#) \*players, [board\\_t](#) board, unsigned int nPawn, unsigned int nPI)

- Set possible movements of a given pawn.*
- void `set_moves_pawn` (`player_t` \*players, `board_t` \*board, unsigned int nPI, int nPawn)
- Set possible movements of a given pawn and set mandatory eat.*
- void `reset_moves_pawns` (`player_t` \*players, int nPawn, unsigned int nPI)
- Reset possible movements of a given pawn.*
- unsigned int `check_canMove` (`player_t` \*players, unsigned int nPed, unsigned int nPI)
- Check if a pawn is able to move.*
- unsigned int `check_while` (`player_t` \*players, unsigned int nPI, unsigned int nPawn)
- Check if a pawn is selectable.*
- unsigned int `check_string` (char \*str)
- Compare if the a given string equals to a valid movement.*
- unsigned int `char_converter` (`board_t` board, unsigned int r, unsigned int c, unsigned int dim\_label)
- Convert a given char on a label to int.*
- char `int_converter` (int num, unsigned int index)
- Convert a given int to a char.*
- char `uppercase` (char c)
- Uppercase a given char.*
- unsigned int `check_char_color` (char c)
- Check if a given char matches a color.*

## Variables

- struct `board` `board_t`
- struct `point` `point_t`
- struct `pawn` `pawn_t`
- struct `player` `player_t`

### 4.5.1 Detailed Description

#### Author

Diego Passarella, Davide Pasqual, Michelle Ravagnan

#### Version

1.0.2

#### Date

2021-01-17

#### Copyright

Copyright (c) 2021

### 4.5.2 Typedef Documentation

#### 4.5.2.1 coord

```
typedef unsigned int coord
```

Definition at line 13 of file game\_engine.h.

#### 4.5.2.2 dim\_board

```
typedef unsigned int dim_board
```

Definition at line 13 of file game\_engine.h.

#### 4.5.2.3 flag

```
typedef unsigned int flag
```

Definition at line 13 of file game\_engine.h.

### 4.5.3 Function Documentation

#### 4.5.3.1 all\_blocked()

```
unsigned int all_blocked (  
    player_t * players,  
    unsigned int nPl )
```

Check if a player can move.

##### Parameters

<i>players</i>	Pointer to the players array.
<i>nPl</i>	Number of the player.

##### Returns

unsigned int Return 1 if there is at least one pawn can move, 0 else.

Definition at line 344 of file game\_engine.c.

#### 4.5.3.2 can\_eat()

```
int can_eat (
    player_t * players,
    unsigned int num_pawn,
    char * str,
    board_t * board,
    unsigned int nPl )
```

Check enemy pawns in a given direction.

##### Parameters

<i>players</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to check.
<i>str</i>	Direction to check from the player's pawn.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of the player.

##### Returns

int Return enemy pawn number that can be eaten, else negative numbers.

Definition at line 222 of file game\_engine.c.

#### 4.5.3.3 char\_converter()

```
unsigned int char_converter (
    board_t board,
    unsigned int r,
    unsigned int c,
    unsigned int dim_label )
```

Convert a given char on a label to int.

##### Parameters

<i>board</i>	Instance of the board.
<i>r</i>	Number of the row (y).
<i>c</i>	Number of the column (x).
<i>dim_label</i>	Dimension of the label of the pawn.

##### Returns

unsigned int Return the conversion of the char to int.

Definition at line 568 of file game\_engine.c.

#### 4.5.3.4 check\_canMove()

```
unsigned int check_canMove (
    player_t * players,
    unsigned int nPed,
    unsigned int nPl )
```

Check if a pawn is able to move.

##### Parameters

<i>players</i>	Pointer to the players array.
<i>nPed</i>	Number of the pawn to check.
<i>nPl</i>	Number of the player.

##### Returns

unsigned int Return 1 if the pawn can move, 0 otherwise.

Definition at line 528 of file game\_engine.c.

#### 4.5.3.5 check\_char\_color()

```
unsigned int check_char_color (
    char c )
```

Check if a given char matches a color.

##### Parameters

<i>c</i>	Char to check.
----------	----------------

##### Returns

unsigned int Return 1 if the char matches, 2 if the char mathes being uppercase, 0 otherwise.

Definition at line 591 of file game\_engine.c.

#### 4.5.3.6 check\_directions()

```
unsigned int check_directions (
    unsigned int * canMove_arr,
    unsigned int dim_canMove,
    char * str )
```

Check if a given string is setted in the pawn's possible movements.

## Parameters

<i>canMove_arr</i>	Pointer to the canMove array of a pawn.
<i>dim_canMove</i>	Dimension of the canMove array, 4 if the pawn is promoted, 2 else.
<i>str</i>	String of the direction to compare.

## Returns

unsigned int Return 1 if the corresponding string is setted on the canMove array, 0 else.

Definition at line 154 of file game\_engine.c.

#### 4.5.3.7 check\_player()

```
int check_player (
    player_t * players,
    unsigned int x,
    unsigned int y,
    unsigned int nPl )
```

Check if in a given coordinate there is an enemy pawn.

## Parameters

<i>players</i>	Pointer to the players array.
<i>x</i>	Number of the column (x).
<i>y</i>	Number of the row (y).
<i>nPl</i>	Number of the player.

## Returns

int Return -1 if a there isn't any enemy pawn, else the number of the enemy player.

Definition at line 131 of file game\_engine.c.

#### 4.5.3.8 check\_spot()

```
unsigned int check_spot (
    board_t board,
    unsigned int row,
    unsigned int col,
    unsigned int dim_label )
```

Check if in a given coordinate on the board there is a pawn or not.

**Parameters**

<i>board</i>	Instance of the board.
<i>row</i>	Number of the row (y).
<i>col</i>	Number of the column (x).
<i>dim_label</i>	Total number of space stored for the label of the pawn.

**Returns**

unsigned int Return 1 if there is a pawn, 0 else.

Definition at line 92 of file game\_engine.c.

**4.5.3.9 check\_string()**

```
unsigned int check_string (  
    char * str )
```

Compare if the a given string equals to a valid movement.

**Parameters**

<i>str</i>	String to compare.
------------	--------------------

**Returns**

unsigned int Return 1 if the string correspond, 0 otherwise.

Definition at line 551 of file game\_engine.c.

**4.5.3.10 check\_while()**

```
unsigned int check_while (  
    player_t * players,  
    unsigned int nPl,  
    unsigned int nPawn )
```

Check if a pawn is selectable.

**Parameters**

<i>players</i>	Pointer to the players array.
<i>nPl</i>	Number of the player.
<i>nPawn</i>	Number of the pawn to check.



**Returns**

unsigned int Return 1 if the pawn is selectable, 0 otherwise.

Definition at line 543 of file game\_engine.c.

**4.5.3.11 initialize\_board()**

```
void initialize_board (
    board_t * board,
    unsigned int cifre )
```

Set char on the board.

'#' white cell, it's where pawns can move ' ' black cell, it's where pawns cannot move

**Parameters**

<i>board</i>	Pointer to the board.
<i>cifre</i>	Number of columns for the label.

Definition at line 21 of file game\_engine.c.

**4.5.3.12 int\_converter()**

```
char int_converter (
    int num,
    unsigned int index )
```

Convert a given int to a char.

**Parameters**

<i>num</i>	Number to be converted.
<i>index</i>	Index of the pow of 10.

**Returns**

char Return the conversion of the int to a char.

Definition at line 577 of file game\_engine.c.

#### 4.5.3.13 is\_empty()

```
int is_empty (
    player_t p )
```

Check if in the player's array there is an unused pawn.

##### Parameters

<i>p</i>	Instance of a player.
----------	-----------------------

##### Returns

int Return the position of the first unused pawn in the array, -1 else.

Definition at line 384 of file game\_engine.c.

#### 4.5.3.14 is\_in()

```
unsigned int is_in (
    int r,
    int c,
    board_t board )
```

Check if a given coordinate are included in the board.

##### Parameters

<i>r</i>	Number of the row (y).
<i>c</i>	Number of the column (x).
<i>board</i>	Instance of the board.

##### Returns

unsigned int Return 1 if the coordinate are inside the board, 0 else.

Definition at line 146 of file game\_engine.c.

#### 4.5.3.15 is\_notstuck()

```
unsigned int is_notstuck (
    player_t * players,
    board_t board,
    unsigned int nPawn,
    unsigned int nPl )
```

Set possible movements of a given pawn.

## Parameters

<i>players</i>	Pointer to the players array.
<i>board</i>	Instance of the board.
<i>nPawn</i>	Number of the pawn to set.
<i>nPl</i>	Number of the player.

## Returns

unsigned int Return 1 if the function runs succesfully, 0 else.

Definition at line 420 of file game\_engine.c.

**4.5.3.16 is\_selected()**

```
unsigned int is_selected (  
    player_t * player,  
    int num_pawn,  
    unsigned int nPl )
```

Check if a given pawn exists in the board.

## Parameters

<i>player</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to check.
<i>nPl</i>	Number of the player.

## Returns

unsigned int Return 1 if the pawn exists in the board, 0 else.

Definition at line 104 of file game\_engine.c.

**4.5.3.17 is\_victory()**

```
unsigned int is_victory (  
    player_t * players )
```

Check if there is a winner.

## Parameters

<i>players</i>	Pointer to the players array.
----------------	-------------------------------

**Returns**

unsigned int Return 1 if player1 won, 2 if player2 won, 17 else.

Definition at line 354 of file game\_engine.c.

**4.5.3.18 max\_pawns()**

```
unsigned int max_pawns (  
    unsigned int r,  
    unsigned int c )
```

Calculate the max number of pawns in a given space.

**Parameters**

<i>r</i>	Total number of rows.
<i>c</i>	Total number of columns.

**Returns**

unsigned int Return the max number of pawns available.

Definition at line 394 of file game\_engine.c.

**4.5.3.19 must\_eat()**

```
void must_eat (  
    player_t * players,  
    board_t * board,  
    unsigned int n_pawn,  
    unsigned int nPl )
```

Set mandatory eat if possible.

**Parameters**

<i>players</i>	Pointer to the players array.
<i>board</i>	Pointer to the board.
<i>n_pawn</i>	Number of the pawn to check.
<i>nPl</i>	Number of the player.

Definition at line 183 of file game\_engine.c.

#### 4.5.3.20 pawn\_promotion()

```
void pawn_promotion (
    player_t * players,
    unsigned int num_pawn,
    unsigned int nPl,
    unsigned int last_row )
```

Promote a given pawn.

##### Parameters

<i>players</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to promote.
<i>nPl</i>	Number of the player.
<i>last_row</i>	Check if a player pawn reaches the end of the board from the perspective of the player.

Definition at line 122 of file game\_engine.c.

#### 4.5.3.21 print\_directions()

```
void print_directions (
    unsigned int * arr,
    unsigned int dim,
    unsigned int np )
```

Print hints for the movements of a given pawn.

##### Parameters

<i>arr</i>	Array of the available movements of a pawn
<i>dim</i>	Dimension of the array 'arr', 4 if the pawn is promoted, 2 if the pawn isn't promoted.
<i>np</i>	Number of the pawn.

Definition at line 44 of file game\_engine.c.

#### 4.5.3.22 remove\_pawn()

```
void remove_pawn (
    board_t * board,
    unsigned int row,
    unsigned int col,
    unsigned int dim_label )
```

Reset label from a given coordinate.

**Parameters**

<i>board</i>	Pointer to the board.
<i>row</i>	Number of the row (y).
<i>col</i>	Number of the column (x).
<i>dim_label</i>	Total number of space stored for the label of the pawn.

Definition at line 112 of file game\_engine.c.

**4.5.3.23 reset\_moves\_pawns()**

```
void reset_moves_pawns (
    player_t * players,
    int nPawn,
    unsigned int nPl )
```

Reset possible movements of a given pawn.

**Parameters**

<i>players</i>	Pointer to the players array.
<i>nPawn</i>	If -1 reset movements of all pawns of a player, else reset movements of the given pawn.
<i>nPl</i>	Number of the player.

Definition at line 501 of file game\_engine.c.

**4.5.3.24 set\_moves\_pawn()**

```
void set_moves_pawn (
    player_t * players,
    board_t * board,
    unsigned int nPl,
    int nPawn )
```

Set possible movements of a given pawn and set mandatory eat.

Calls to is\_notstuck, must\_eat and reset\_moves\_pawns.

**Parameters**

<i>players</i>	Pointer to the players array.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of the player.
<i>nPawn</i>	If -1 set movements of all pawns of a player, else set movements of the given pawn.

Definition at line 477 of file game\_engine.c.

#### 4.5.3.25 update\_board()

```
void update_board (
    board_t * t,
    player_t * player )
```

Update the board after a player move.

##### Parameters

<i>t</i>	Pointer to the board.
<i>player</i>	Pointer of players array.

Definition at line 74 of file game\_engine.c.

#### 4.5.3.26 uppercase()

```
char uppercase (
    char c )
```

Uppercase a given char.

##### Parameters

<i>c</i>	Char to be uppercased.
----------	------------------------

##### Returns

char Return the uppercased char.

Definition at line 587 of file game\_engine.c.

### 4.5.4 Variable Documentation

#### 4.5.4.1 board\_t

```
struct board board_t
```

#### 4.5.4.2 pawn\_t

```
struct pawn pawn_t
```

#### 4.5.4.3 player\_t

```
struct player player_t
```

#### 4.5.4.4 point\_t

```
struct point point_t
```

## 4.6 src/ia/ia.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "../colors/colors.h"
#include "../game_engine/game_engine.h"
#include "ia.h"
#include "../memory_management/memory_management.h"
#include "../movement/movement.h"
#include "../user_interaction/user_interaction.h"
```

## Data Structures

- struct [valueMinimax](#)

## Functions

- unsigned int [round\\_ia\\_random](#) ([player\\_t](#) \*players, [board\\_t](#) \*board, unsigned int nPI)  
*Move the player randomly.*
- int [last\\_move](#) ([player\\_t](#) \*players, unsigned int nPI)  
*Simplify the pawn selection in the random when there is only 1 pawn left.*
- int [round\\_ia\\_minimax](#) ([player\\_t](#) \*players, [board\\_t](#) \*board, unsigned int nPI, unsigned int depth)  
*Move the player using the minimax algorithm.*
- int [minimax](#) ([board\\_t](#) board, [player\\_t](#) \*players, int depth, unsigned int nPed, unsigned int nPI, [valueMinimax\\_t](#) \*v, int cheat, int \*alpha, int \*beta)  
*Algorithm which computes the best move possible for a player.*



- int [call\\_minimax](#) ([board\\_t](#) \*board\_copy, [player\\_t](#) \*players\_copy, unsigned int depth, unsigned int nPed, unsigned int nPl, [valueMinimax\\_t](#) \*v, char \*str, int maxEval, int cheat, int \*alpha, int \*beta)  
*Synthesizes some operations in the minimax.*
- unsigned int [interrupt\\_minimax](#) (int \*alpha, int \*beta)  
*Compare alpha and beta.*
- int [evaluate\\_score](#) ([board\\_t](#) board, [player\\_t](#) \*players)  
*Calculate the score of the board.*
- void [print\\_minimax](#) ([valueMinimax\\_t](#) \*value\_arr, unsigned int dim\_arr)  
*Print minimax results.*
- unsigned int [max](#) ([valueMinimax\\_t](#) \*arr, unsigned int dim)  
*Calculate the max value among all previous results.*
- void [destroy\\_value\\_minimax](#) ([valueMinimax\\_t](#) \*arr, unsigned int dim)  
*Deallocate all allocated memory by round\_ia\_minimax.*

## 4.6.1 Function Documentation

### 4.6.1.1 call\_minimax()

```
int call_minimax (
    board\_t * board_copy,
    player\_t * players_copy,
    unsigned int depth,
    unsigned int nPed,
    unsigned int nPl,
    valueMinimax\_t * v,
    char * str,
    int maxEval,
    int cheat,
    int * alpha,
    int * beta )
```

Synthesizes some operations in the minimax.

#### Parameters

<i>board_copy</i>	Pointer to a copy of a board.
<i>players_copy</i>	Pointer to a copy of players array.
<i>depth</i>	Depth of the tree generated by the minimax.
<i>nPed</i>	Number of the pawn to calculate the move.
<i>nPl</i>	Number of the player.
<i>v</i>	Pointer to an element of the <a href="#">valueMinimax</a> array.
<i>str</i>	Direction considered by the minimax.
<i>maxEval</i>	Compare the score of this call with the others.
<i>cheat</i>	Determines the selection of the min value or max value in that level of the graph.
<i>alpha</i>	Min value of the player 1 can reach.
<i>beta</i>	Max value of the player 2 can reach.

**Returns**

int Return the max value or the min value based on the cheat flag.

Definition at line 257 of file ia.c.

**4.6.1.2 destroy\_value\_minimax()**

```
void destroy_value_minimax (
    valueMinimax_t * arr,
    unsigned int dim )
```

Deallocate all allocated memory by round\_ia\_minimax.

**Parameters**

<i>arr</i>	Pointer to <a href="#">valueMinimax</a> array.
<i>dim</i>	Dimension of the <a href="#">valueMinimax</a> array.

Definition at line 356 of file ia.c.

**4.6.1.3 evaluate\_score()**

```
int evaluate_score (
    board_t board,
    player_t * players )
```

Calculate the score of the board.

**Parameters**

<i>board</i>	Instance of the board.
<i>players</i>	Pointer to players array.

**Returns**

int Return the score.

Definition at line 308 of file ia.c.

**4.6.1.4 interrupt\_minimax()**

```
unsigned int interrupt_minimax (
    int * alpha,
    int * beta )
```

Compare alpha and beta.

**Parameters**

<i>alpha</i>	Min value of the player 1 can reach.
<i>beta</i>	Max value of the player 2 can reach.

**Returns**

unsigned int Return 0 if alpha or beta are inf or alpha < beta, 1 otherwise.

Definition at line 301 of file ia.c.

**4.6.1.5 last\_move()**

```
int last_move (
    player_t * players,
    unsigned int nPl )
```

Simplify the pawn selection in the random when there is only 1 pawn left.

**Parameters**

<i>players</i>	Pointer to players array.
<i>nPl</i>	Number of the player to move.

**Returns**

int Return the pawn number, -1 otherwise.

Definition at line 72 of file ia.c.

**4.6.1.6 max()**

```
unsigned int max (
    valueMinimax_t * arr,
    unsigned int dim )
```

Calculate the max value among all previous results.

**Parameters**

<i>arr</i>	Pointer to <a href="#">valueMinimax</a> array.
<i>dim</i>	Dimension of the <a href="#">valueMinimax</a> array.

### Returns

unsigned int Return the index of the array containing the max value.

Definition at line 341 of file ia.c.

#### 4.6.1.7 minimax()

```
int minimax (
    board_t board,
    player_t * players,
    int depth,
    unsigned int nPed,
    unsigned int nPl,
    valueMinimax_t * v,
    int cheat,
    int * alpha,
    int * beta )
```

Algorithm which computes the best move possible for a player.

### Parameters

<i>board</i>	Instance of the board.
<i>players</i>	Pointer to players array.
<i>depth</i>	Depth of the tree generated by the minimax.
<i>nPed</i>	Number of the pawn to calculate the move.
<i>nPl</i>	Number of the player.
<i>v</i>	Pointer to an element of the <a href="#">valueMinimax</a> array.
<i>cheat</i>	Determines the selection of the min value or max value in that level of the graph.
<i>alpha</i>	Min value of the player 1 can reach.
<i>beta</i>	Max value of the player 2 can reach.

### Returns

int Return the evaluated score.

Definition at line 125 of file ia.c.

#### 4.6.1.8 print\_minimax()

```
void print_minimax (
    valueMinimax_t * value_arr,
    unsigned int dim_arr )
```

Print minimax results.

**Parameters**

<i>value_arr</i>	Pointer to <a href="#">valueMinimax</a> array.
<i>dim_arr</i>	Dimension of the <a href="#">valueMinimax</a> array.

Definition at line 330 of file ia.c.

**4.6.1.9 round\_ia\_minimax()**

```
int round_ia_minimax (
    player\_t * players,
    board\_t * board,
    unsigned int nPl,
    unsigned int depth )
```

Move the player using the minimax algorithm.

**Parameters**

<i>players</i>	Pointer to players array.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of the player to move.
<i>depth</i>	Depth of the tree generated by the minimax.

**Returns**

int Return 2 if there are errors, 4 otherwise.

Definition at line 87 of file ia.c.

**4.6.1.10 round\_ia\_random()**

```
unsigned int round_ia_random (
    player\_t * players,
    board\_t * board,
    unsigned int nPl )
```

Move the player randomly.

**Parameters**

<i>players</i>	Pointer to players array.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of the player to move.

**Returns**

unsigned int Return 2 if there are errors, 4 otherwise.

Definition at line 21 of file ia.c.

## 4.7 src/ia/ia.h File Reference

**Typedefs**

- typedef struct [valueMinimax](#) [valueMinimax\\_t](#)  
*Auxiliary struct for the minimax.*

**Functions**

- unsigned int [round\\_ia\\_random](#) ([player\\_t](#) \*players, [board\\_t](#) \*board, unsigned int nPI)  
*Move the player randomly.*
- int [last\\_move](#) ([player\\_t](#) \*players, unsigned int nPI)  
*Simplify the pawn selection in the random when there is only 1 pawn left.*
- int [round\\_ia\\_minimax](#) ([player\\_t](#) \*players, [board\\_t](#) \*board, unsigned int nPI, unsigned int depth)  
*Move the player using the minimax algorithm.*
- int [minimax](#) ([board\\_t](#) board, [player\\_t](#) \*players, int depth, unsigned int nPed, unsigned int nPI, [valueMinimax\\_t](#) \*v, int cheat, int \*alpha, int \*beta)  
*Algorithm which computes the best move possible for a player.*
- int [call\\_minimax](#) ([board\\_t](#) \*board\_copy, [player\\_t](#) \*players\_copy, unsigned int depth, unsigned int nPed, unsigned int nPI, [valueMinimax\\_t](#) \*v, char \*str, int maxEval, int cheat, int \*alpha, int \*beta)  
*Synthesizes some operations in the minimax.*
- unsigned int [interrupt\\_minimax](#) (int \*alpha, int \*beta)  
*Compare alpha and beta.*
- int [evaluate\\_score](#) ([board\\_t](#) board, [player\\_t](#) \*players)  
*Calculate the score of the board.*
- void [print\\_minimax](#) ([valueMinimax\\_t](#) \*value\_arr, unsigned int dim\_arr)  
*Print minimax results.*
- unsigned int [max](#) ([valueMinimax\\_t](#) \*arr, unsigned int dim)  
*Calculate the max value among all previous results.*
- void [destroy\\_value\\_minimax](#) ([valueMinimax\\_t](#) \*arr, unsigned int dim)  
*Deallocate all allocated memory by round\_ia\_minimax.*

### 4.7.1 Detailed Description

**Author**

Diego Passarella, Davide Pasqual, Michelle Ravagnan

**Version**

1.0.2

**Date**

2021-01-17

**Copyright**

Copyright (c) 2021

## 4.7.2 Typedef Documentation

### 4.7.2.1 valueMinimax\_t

```
typedef struct valueMinimax valueMinimax_t
```

Auxiliary struct for the minimax.

In order to improve the efficiency of the minimax function we preferred to create a new struct instead of calling another function similar to minimax.

#### Parameters

<i>value</i>	Evaluated score of the minimax or -9017 if the pawn is blocked.
<i>directions</i>	Array containing the string of the direction.

Definition at line 1 of file ia.h.

## 4.7.3 Function Documentation

### 4.7.3.1 call\_minimax()

```
int call_minimax (
    board_t * board_copy,
    player_t * players_copy,
    unsigned int depth,
    unsigned int nPed,
    unsigned int nPl,
    valueMinimax_t * v,
    char * str,
    int maxEval,
    int cheat,
    int * alpha,
    int * beta )
```

Synthesizes some operations in the minimax.

#### Parameters

<i>board_copy</i>	Pointer to a copy of a board.
<i>players_copy</i>	Pointer to a copy of players array.
<i>depth</i>	Depth of the tree generated by the minimax.
<i>nPed</i>	Number of the pawn to calculate the move.
<i>nPl</i>	Number of the player.
<i>v</i>	Pointer to an element of the <a href="#">valueMinimax</a> array.



## Parameters

<i>str</i>	Direction considered by the minimax.
<i>maxEval</i>	Compare the score of this call with the others.
<i>cheat</i>	Determines the selection of the min value or max value in that level of the graph.
<i>alpha</i>	Min value of the player 1 can reach.
<i>beta</i>	Max value of the player 2 can reach.

## Returns

int Return the max value or the min value based on the cheat flag.

Definition at line 257 of file ia.c.

### 4.7.3.2 destroy\_value\_minimax()

```
void destroy_value_minimax (
    valueMinimax_t * arr,
    unsigned int dim )
```

Deallocate all allocated memory by round\_ia\_minimax.

## Parameters

<i>arr</i>	Pointer to <a href="#">valueMinimax</a> array.
<i>dim</i>	Dimension of the <a href="#">valueMinimax</a> array.

Definition at line 356 of file ia.c.

### 4.7.3.3 evaluate\_score()

```
int evaluate_score (
    board_t board,
    player_t * players )
```

Calculate the score of the board.

## Parameters

<i>board</i>	Instance of the board.
<i>players</i>	Pointer to players array.

**Returns**

int Return the score.

Definition at line 308 of file ia.c.

**4.7.3.4 interrupt\_minimax()**

```
unsigned int interrupt_minimax (
    int * alpha,
    int * beta )
```

Compare alpha and beta.

**Parameters**

<i>alpha</i>	Min value of the player 1 can reach.
<i>beta</i>	Max value of the player 2 can reach.

**Returns**

unsigned int Return 0 if alpha or beta are inf or alpha < beta, 1 otherwise.

Definition at line 301 of file ia.c.

**4.7.3.5 last\_move()**

```
int last_move (
    player_t * players,
    unsigned int nPl )
```

Simplify the pawn selection in the random when there is only 1 pawn left.

**Parameters**

<i>players</i>	Pointer to players array.
<i>nPl</i>	Number of the player to move.

**Returns**

int Return the pawn number, -1 otherwise.

Definition at line 72 of file ia.c.

#### 4.7.3.6 max()

```
unsigned int max (
    valueMinimax_t * arr,
    unsigned int dim )
```

Calculate the max value among all previous results.

##### Parameters

<i>arr</i>	Pointer to <a href="#">valueMinimax</a> array.
<i>dim</i>	Dimension of the <a href="#">valueMinimax</a> array.

##### Returns

unsigned int Return the index of the array containing the max value.

Definition at line 341 of file ia.c.

#### 4.7.3.7 minimax()

```
int minimax (
    board_t board,
    player_t * players,
    int depth,
    unsigned int nPed,
    unsigned int nPl,
    valueMinimax_t * v,
    int cheat,
    int * alpha,
    int * beta )
```

Algorithm which computes the best move possible for a player.

##### Parameters

<i>board</i>	Instance of the board.
<i>players</i>	Pointer to players array.
<i>depth</i>	Depth of the tree generated by the minimax.
<i>nPed</i>	Number of the pawn to calculate the move.
<i>nPl</i>	Number of the player.
<i>v</i>	Pointer to an element of the <a href="#">valueMinimax</a> array.
<i>cheat</i>	Determines the selection of the min value or max value in that level of the graph.
<i>alpha</i>	Min value of the player 1 can reach.
<i>beta</i>	Max value of the player 2 can reach.

**Returns**

int Return the evaluated score.

Definition at line 125 of file ia.c.

**4.7.3.8 print\_minimax()**

```
void print_minimax (
    valueMinimax_t * value_arr,
    unsigned int dim_arr )
```

Print minimax results.

**Parameters**

<i>value_arr</i>	Pointer to <a href="#">valueMinimax</a> array.
<i>dim_arr</i>	Dimension of the <a href="#">valueMinimax</a> array.

Definition at line 330 of file ia.c.

**4.7.3.9 round\_ia\_minimax()**

```
int round_ia_minimax (
    player_t * players,
    board_t * board,
    unsigned int nPl,
    unsigned int depth )
```

Move the player using the minimax algorithm.

**Parameters**

<i>players</i>	Pointer to players array.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of the player to move.
<i>depth</i>	Depth of the tree generated by the minimax.

**Returns**

int Return 2 if there are errors, 4 otherwise.

Definition at line 87 of file ia.c.

#### 4.7.3.10 round\_ia\_random()

```
unsigned int round_ia_random (
    player_t * players,
    board_t * board,
    unsigned int nPl )
```

Move the player randomly.

##### Parameters

<i>players</i>	Pointer to players array.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of the player to move.

##### Returns

unsigned int Return 2 if there are errors, 4 otherwise.

Definition at line 21 of file ia.c.

## 4.8 src/memory\_management/memory\_management.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "../colors/colors.h"
#include "../game_engine/game_engine.h"
#include "../ia/ia.h"
#include "memory_management.h"
#include "../movement/movement.h"
#include "../user_interaction/user_interaction.h"
```

### Functions

- **board\_t \* create\_board** (unsigned int n\_rows, unsigned int n\_cols, unsigned int cifre)  
*Create a board object.*
- **player\_t \* create\_pawns** (unsigned int totPawns, char player1, char player2, unsigned int cifre, **board\_t board**)  
*Create a pawns object.*
- **board\_t \* copy\_board** (**board\_t board**, **board\_t \*newBoard**, unsigned int cifre, unsigned int set)  
*It copy all value of board to another board, eventually allocating memory if necessary.*
- **player\_t \* player\_copy** (**player\_t \*players**, **player\_t \*newPlayers**, unsigned int dim\_label, unsigned int set)  
*It copy all value of array players to another array players, eventually allocating memory if necessary.*
- void **destroy\_player** (**player\_t \*players**)  
*Deallocate all allocated memory by create\_pawns.*
- void **destroy\_board** (**board\_t \*board**)  
*Deallocate all allocated memory by create\_board.*
- void **restore\_copy** (**board\_t board**, **player\_t \*players**, **board\_t \*board\_copy**, **player\_t \*players\_copy**)  
*Calls the two copy functions to summarize the various copy calls into one.*

## 4.8.1 Function Documentation

### 4.8.1.1 copy\_board()

```
board_t* copy_board (
    board_t board,
    board_t * newBoard,
    unsigned int cifre,
    unsigned int set )
```

It copy all value of board to another board,eventually allocating memory if necessary.

#### Parameters

<i>board</i>	Instance of a board.
<i>newBoard</i>	Pointer of a board where do the copy.
<i>cifre</i>	Number of columns that are necessary to represent a label of one pawn.
<i>set</i>	If 1 then the fuction allocating memory,0 else.

#### Returns

*board\_t\** Return a pointer of a board that it is the copy of chessboard.

Definition at line 146 of file memory\_management.c.

### 4.8.1.2 create\_board()

```
board_t* create_board (
    unsigned int n_rows,
    unsigned int n_cols,
    unsigned int cifre )
```

Create a board object.

#### Parameters

<i>n_rows</i>	Number of row of the new board.
<i>n_cols</i>	Number of columns of the new board.
<i>cifre</i>	Number of columns that are necessary to represent a label of one pawn.

#### Returns

*board\_t\** Return a pointer of the board.

Definition at line 17 of file memory\_management.c.

#### 4.8.1.3 create\_pawns()

```
player_t* create_pawns (
    unsigned int totPawns,
    char player1,
    char player2,
    unsigned int cifre,
    board_t board )
```

Create a pawns object.

##### Parameters

<i>totPawns</i>	Number of pawns for each player.
<i>player1</i>	Characters of the pawn of player 1, and this is helpful for the <code>print_board("../game_engine/game_engine.h")</code> .
<i>player2</i>	Characters of the pawn of player 2, and this is helpful for the <code>print_board("../game_engine/game_engine.h")</code> .
<i>cifre</i>	Numbers of characters to represent the number of a pawn.
<i>board</i>	Instance of the board.

##### Returns

`player_t*` Return a pointer of the array players.

Definition at line 37 of file `memory_management.c`.

#### 4.8.1.4 destroy\_board()

```
void destroy_board (
    board_t * board )
```

Deallocate all allocated memory by `create_board`.

##### Parameters

<i>board</i>	Pointer of a board.
--------------	---------------------

Definition at line 254 of file `memory_management.c`.

#### 4.8.1.5 destroy\_player()

```
void destroy_player (
    player_t * players )
```

Deallocate all allocated memory by `create_pawns`.

**Parameters**

<i>players</i>	Pointer of players array.
----------------	---------------------------

Definition at line 237 of file memory\_management.c.

**4.8.1.6 player\_copy()**

```
player_t* player_copy (
    player_t * players,
    player_t * newPlayers,
    unsigned int dim_label,
    unsigned int set )
```

It copy all value of array players to another array players, eventually allocating memory if necessary.

**Parameters**

<i>players</i>	Pointer of players array.
<i>newPlayers</i>	Pointer of players array where to make the copy.
<i>dim_label</i>	Dimension of label array of pawn.
<i>set</i>	If 1 then the fuction allocating memory, 0 else.

**Returns**

*player\_t\** Return a pointer of a players array that is the copy of players parameter.

Definition at line 177 of file memory\_management.c.

**4.8.1.7 restore\_copy()**

```
void restore_copy (
    board_t board,
    player_t * players,
    board_t * board_copy,
    player_t * players_copy )
```

Calls the two copy functions to summarize the various copy calls into one.

**Parameters**

<i>board</i>	Instance of a board.
<i>players</i>	Pointer of players array.
<i>board_copy</i>	Pointer of a board where do the copy.
<i>players_copy</i>	Pointer of players array where to make the copy.



Definition at line 264 of file memory\_management.c.

## 4.9 src/memory\_management/memory\_management.h File Reference

### Functions

- `board_t * create_board` (unsigned int n\_rows, unsigned int n\_cols, unsigned int cifre)  
*Create a board object.*
- `player_t * create_pawns` (unsigned int totPawns, char player1, char player2, unsigned int cifre, `board_t board`)  
*Create a pawns object.*
- `board_t * copy_board` (`board_t board`, `board_t *newBoard`, unsigned int cifre, unsigned int set)  
*It copy all value of board to another board,eventually allocating memory if necessary.*
- `player_t * player_copy` (`player_t *players`, `player_t *newPlayers`, unsigned int dim\_label, unsigned int set)  
*It copy all value of array players to another array players,eventually allocating memory if necessary.*
- void `destroy_player` (`player_t *players`)  
*Deallocate all allocated memory by create\_pawns.*
- void `destroy_board` (`board_t *board`)  
*Deallocate all allocated memory by create\_board.*
- void `restore_copy` (`board_t board`, `player_t *players`, `board_t *board_copy`, `player_t *players_copy`)  
*Calls the two copy functions to summarize the various copy calls into one.*

### 4.9.1 Detailed Description

#### Author

Diego Passarella, Davide Pasqual, Michelle Ravagnan

#### Version

1.0.2

#### Date

2021-01-17

#### Copyright

Copyright (c) 2021

### 4.9.2 Function Documentation

#### 4.9.2.1 copy\_board()

```
board_t* copy_board (
    board_t board,
    board_t * newBoard,
    unsigned int cifre,
    unsigned int set )
```

It copy all value of board to another board,eventually allocating memory if necessary.

**Parameters**

<i>board</i>	Instance of a board.
<i>newBoard</i>	Pointer of a board where do the copy.
<i>cifre</i>	Number of columns that are necessary to represent a label of one pawn.
<i>set</i>	If 1 then the fuction allocating memory,0 else.

**Returns**

`board_t*` Return a pointer of a board that it is the copy of chessboard.

Definition at line 146 of file `memory_management.c`.

**4.9.2.2 create\_board()**

```
board_t* create_board (
    unsigned int n_rows,
    unsigned int n_cols,
    unsigned int cifre )
```

Create a board object.

**Parameters**

<i>n_rows</i>	Number of row of the new board.
<i>n_cols</i>	Number of columns of the new board.
<i>cifre</i>	Number of columns that are necessary to represent a label of one pawn.

**Returns**

`board_t*` Return a pointer of the board.

Definition at line 17 of file `memory_management.c`.

**4.9.2.3 create\_pawns()**

```
player_t* create_pawns (
    unsigned int totPawns,
    char player1,
    char player2,
    unsigned int cifre,
    board_t board )
```

Create a pawns object.

**Parameters**

<i>totPawns</i>	Number of pawns for each player.
<i>player1</i>	Characters of the pawn of player 1,and this is helpful for the <code>print_board("../game_engine/game_engine.h")</code> .
<i>player2</i>	Characters of the pawn of player 2,and this is helpful for the <code>print_board("../game_engine/game_engine.h")</code> .
<i>cifre</i>	Numbers of characters to represent the number of a pawn.
<i>board</i>	Instance of the board.

**Returns**

`player_t*` Return a pointer of the array players.

Definition at line 37 of file `memory_management.c`.

**4.9.2.4 destroy\_board()**

```
void destroy_board (  
    board_t * board )
```

Deallocate all allocated memory by `create_board`.

**Parameters**

<i>board</i>	Pointer of a board.
--------------	---------------------

Definition at line 254 of file `memory_management.c`.

**4.9.2.5 destroy\_player()**

```
void destroy_player (  
    player_t * players )
```

Deallocate all allocated memory by `create_pawns`.

**Parameters**

<i>players</i>	Pointer of players array.
----------------	---------------------------

Definition at line 237 of file `memory_management.c`.

#### 4.9.2.6 player\_copy()

```
player_t* player_copy (
    player_t * players,
    player_t * newPlayers,
    unsigned int dim_label,
    unsigned int set )
```

It copy all value of array players to another array players, eventually allocating memory if necessary.

##### Parameters

<i>players</i>	Pointer of players array.
<i>newPlayers</i>	Pointer of players array where to make the copy.
<i>dim_label</i>	Dimension of label array of pawn.
<i>set</i>	If 1 then the fuction allocating memory, 0 else.

##### Returns

*player\_t\** Return a pointer of a players array that is the copy of players parameter.

Definition at line 177 of file memory\_management.c.

#### 4.9.2.7 restore\_copy()

```
void restore_copy (
    board_t board,
    player_t * players,
    board_t * board_copy,
    player_t * players_copy )
```

Calls the two copy functions to summarize the various copy calls into one.

##### Parameters

<i>board</i>	Instance of a board.
<i>players</i>	Pointer of players array.
<i>board_copy</i>	Pointer of a board where do the copy.
<i>players_copy</i>	Pointer of players array where to make the copy.

Definition at line 264 of file memory\_management.c.

## 4.10 src/movement/movement.c File Reference

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <math.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "../colors/colors.h"
#include "../game_engine/game_engine.h"
#include "../ia/ia.h"
#include "../memory_management/memory_management.h"
#include "movement.h"
#include "../user_interaction/user_interaction.h"
```

## Functions

- unsigned int `move_noeat` (`player_t` \*players, unsigned int num\_pawn, char \*str, `board_t` \*board, unsigned int nPl)  
*Move a given number of pawn without eat to a string direction.*
- unsigned int `eat` (`player_t` \*players, char \*str, unsigned int num\_pawn, `board_t` board, unsigned int enemy\_pawn, unsigned int nPl)  
*Eat the enemy pawn respect us.*
- int `move_p1` (`player_t` \*players, unsigned int num\_pawn, char \*str, `board_t` \*board, unsigned int nPl)  
*This is the main function that allows the pawn to move in the 4 available directions.*
- int `move_p2` (`player_t` \*players, unsigned int num\_pawn, char \*str, `board_t` \*board)  
*Move a given number of pawn of the second player.*

### 4.10.1 Function Documentation

#### 4.10.1.1 `eat()`

```
unsigned int eat (
    player_t * players,
    char * str,
    unsigned int num_pawn,
    board_t board,
    unsigned int enemy_pawn,
    unsigned int nPl )
```

Eat the enemy pawn respect us.

There are 4 string directions :

- > "l" which means left respect the num\_pawn pawn.
- > "r" which means right respect the num\_pawn pawn.
- > "botr" which means bottom right respect the num\_pawn pawn.
- > "botl" which means bottom left respect the num\_pawn pawn.

There are 5 steps :

- > 1 step : calculate the number of enemy\_player and save with a char array the new characters of the num\_pawn(it has length 3 due to the height limitation of the pawn).
- > 2 step : there are 2 main cases that will be analyzed below.
- > 3 step : modify the label of num\_pawn pawn (if his height < 3), then modify the other fields such as the new coordinates(according to the string direction).
- > 4 step : remove the pawn at the old coordinates on the board.
- > 5 step : eventually promote the num\_pawn pawn.

Main cases of 2 step :

- > 1 case : the enemy pawn has grade equal to 1 so it just set it to zero to remove it from the board when the update\_board("../game\_engine/game\_engine.h") is done.
- > 2 case : mainly it concerns when the opponent's pawn includes more pieces and we distinguish two main sub-cases :
  - > 1 sub cases : when the highest opponent pawn in the tower is the same as the next one, simply the control of the new opponent pawn is maintained by the opposing player who is deprived of the highest pawn of the tower.
  - > 2 sub cases : when the highest pawn of the opposing tower is different from the next one, control of the pawn is lost and attributed to the other player.

#### Parameters

<i>players</i>	Pointer to the players array.
<i>str</i>	String of direction where to move the pawn.
<i>num_pawn</i>	Number of the pawn to move and eat the enemy pawn.
<i>board</i>	Pointer to the board.
<i>enemy_pawn</i>	Number of the enemy pawn that will be eaten.
<i>nPl</i>	Number of player of num_pawn.

#### Returns

unsigned int

Definition at line 71 of file movement.c.

#### 4.10.1.2 move\_noeat()

```
unsigned int move_noeat (
    player_t * players,
    unsigned int num_pawn,
    char * str,
    board_t * board,
    unsigned int nPl )
```

Move a given number of pawn without eat to a string direction.

There are 4 steps :

- > 1 step : check with "strcmp" function( string.h library)if the given string is correct.
- > 2 step : check if in that direction there isn't any pawn and the new coordinates of the pawn are inside the chessboard.

- > 3 step : assign new coordinates to the pawn and remove the pawn at the old coordinates on the board.
- > 4 step : eventually promote the num\_pawn pawn.

There are 4 string directions :

- > "l" which means left respect the num\_pawn pawn.
- > "r" which means right respect the num\_pawn pawn.
- > "botr" which means bottom right respect the num\_pawn pawn.
- > "botl" which means bottom left respect the num\_pawn pawn.

#### Parameters

<i>players</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to move.
<i>str</i>	String of direction where to move the pawn.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of player of num_pawn.

#### Returns

unsigned int Return 1 if the pawn moves correctly,0 else.

Definition at line 16 of file movement.c.

#### 4.10.1.3 move\_p1()

```
int move_p1 (
    player_t * players,
    unsigned int num_pawn,
    char * str,
    board_t * board,
    unsigned int nPl )
```

This is the main function that allows the pawn to move in the 4 available directions.

#### Parameters

<i>players</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to move.
<i>str</i>	String of direction where to move the pawn.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of player of num_pawn.

#### Returns

int Return a number  $\geq 0$  if the num\_pawn ate and matches to the number of enemy pawn that it was eaten, -1 if the num\_pawn move without eat, -2 if did nothing , -4 if there is an error in can\_eat function(#include "../game\_engine/game\_engine.h");

Definition at line 165 of file movement.c.

#### 4.10.1.4 move\_p2()

```
int move_p2 (
    player_t * players,
    unsigned int num_pawn,
    char * str,
    board_t * board )
```

Move a given number of pawn of the second player.

It is very similar to move\_p1 but it is different because the directions of player 2 move inversely with respect to player 1, for example if I have to go in the left direction of player 1, the direction of player 2 is not left, but bottom left. This is due to the fact that in the chessboard player 1 is in the last lines, instead player 2 in the first ones, so if I move to the left of player 1 the y coordinate of the pawn decreases, on the contrary in player 2 that when I have to go on the left the y coordinate increases (as in fact when compared to player 1 I go to the bottom left and then reusing the functions of player 1).

There are 4 string directions :

- > "l" which means left respect the num\_pawn pawn.
- > "r" which means right respect the num\_pawn pawn.
- > "botr" which means bottom right respect the num\_pawn pawn.
- > "botl" which means bottom left respect the num\_pawn pawn.

The 4 directions of player 2 respect player 1 :

- > "l" that respect player 1 it is "botl".
- > "r" that respect player 1 it is "botr".
- > "botr" that respect player 1 it is "r".
- > "l" that respect player 1 it is "l".

##### Parameters

<i>players</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to move.
<i>str</i>	String of direction where to move the pawn.
<i>board</i>	Pointer to the board.

##### Returns

int Return a number  $\geq 0$  if the num\_pawn ate and matches to the number of enemy pawn that it was eaten, -1 if the num\_pawn move without eat, -2 if did nothing , -4 if there is an error in can\_eat function(#include "../game\_engine/game\_engine.h");

Definition at line 182 of file movement.c.



## 4.11 src/movement/movement.h File Reference

### Functions

- unsigned int `move_noeat` (`player_t` \*players, unsigned int num\_pawn, char \*str, `board_t` \*board, unsigned int nPI)  
*Move a given number of pawn without eat to a string direction.*
- unsigned int `eat` (`player_t` \*players, char \*str, unsigned int num\_pawn, `board_t` board, unsigned int enemy↵\_pawn, unsigned int nPI)  
*Eat the enemy pawn respect us.*
- int `move_p1` (`player_t` \*players, unsigned int num\_pawn, char \*str, `board_t` \*board, unsigned int nPI)  
*This is the main function that allows the pawn to move in the 4 available directions.*
- int `move_p2` (`player_t` \*players, unsigned int num\_pawn, char \*str, `board_t` \*board)  
*Move a given number of pawn of the second player.*

### 4.11.1 Detailed Description

#### Author

Diego Passarella, Davide Pasqual, Michelle Ravagnan

#### Version

1.0.2

#### Date

2021-01-17

#### Copyright

Copyright (c) 2021

### 4.11.2 Function Documentation

#### 4.11.2.1 eat()

```
unsigned int eat (
    player_t * players,
    char * str,
    unsigned int num_pawn,
    board_t board,
    unsigned int enemy_pawn,
    unsigned int nPl )
```

Eat the enemy pawn respect us.

There are 4 string directions :

- > "l" which means left respect the num\_pawn pawn.
- > "r" which means right respect the num\_pawn pawn.
- > "botr" which means bottom right respect the num\_pawn pawn.
- > "botl" which means bottom left respect the num\_pawn pawn.

There are 5 steps :

- > 1 step : calculate the number of enemy\_player and save with a char array the new characters of the num\_pawn(it has length 3 due to the height limitation of the pawn).
- > 2 step : there are 2 main cases that will be analyzed below.
- > 3 step : modify the label of num\_pawn pawn (if his height < 3),then modify the other fields such as the new coordinates(according to the string direction).
- > 4 step : remove the pawn at the old coordinates on the board.
- > 5 step : eventually promote the num\_pawn pawn.

Main cases of 2 step :

- > 1 case : the enemy pawn has grade equal to 1 so it just set it to zero to remove it from the board when the update\_board("../game\_engine/game\_engine.h") is done.
- > 2 case : mainly it concerns when the opponent's pawn includes more pieces and we distinguish two main sub-cases :
  - > 1 sub cases : when the highest opponent pawn in the tower is the same as the next one, simply the control of the new opponent pawn is maintained by the opposing player who is deprived of the highest pawn of the tower.
  - > 2 sub cases : when the highest pawn of the opposing tower is different from the next one, control of the pawn is lost and attributed to the other player.

##### Parameters

<i>players</i>	Pointer to the players array.
<i>str</i>	String of direction where to move the pawn.
<i>num_pawn</i>	Number of the pawn to move and eat the enemy pawn.
<i>board</i>	Pointer to the board.
<i>enemy_pawn</i>	Number of the enemy pawn that will be eaten.
<i>nPl</i>	Number of player of num_pawn.

##### Returns

unsigned int

Definition at line 71 of file movement.c.

#### 4.11.2.2 move\_noeat()

```
unsigned int move_noeat (  
    player_t * players,  
    unsigned int num_pawn,  
    char * str,  
    board_t * board,  
    unsigned int nPl )
```

Move a given number of pawn without eat to a string direction.

There are 4 steps :

- > 1 step : check with "strcmp" function( string.h library)if the given string is correct.
- > 2 step : check if in that direction there isn't any pawn and the new coordinates of the pawn are inside the chessboard.
- > 3 step : assign new coordinates to the pawn and remove the pawn at the old coordinates on the board.
- > 4 step : eventually promote the num\_pawn pawn.

There are 4 string directions :

- > "l" which means left respect the num\_pawn pawn.
- > "r" which means right respect the num\_pawn pawn.
- > "botr" which means bottom right respect the num\_pawn pawn.
- > "botl" which means bottom left respect the num\_pawn pawn.

##### Parameters

<i>players</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to move.
<i>str</i>	String of direction where to move the pawn.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of player of num_pawn.

##### Returns

unsigned int Return 1 if the pawn moves correctly,0 else.

Definition at line 16 of file movement.c.

#### 4.11.2.3 move\_p1()

```
int move_p1 (  
    player_t * players,  
    unsigned int num_pawn,  
    char * str,  
    board_t * board,  
    unsigned int nPl )
```

This is the main function that allows the pawn to move in the 4 available directions.

**Parameters**

<i>players</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to move.
<i>str</i>	String of direction where to move the pawn.
<i>board</i>	Pointer to the board.
<i>nPl</i>	Number of player of num_pawn.

**Returns**

int Return a number  $\geq 0$  if the num\_pawn ate and matches to the number of enemy pawn that it was eaten, -1 if the num\_pawn move without eat, -2 if did nothing, -4 if there is an error in can\_eat function(#include "../game\_engine/game\_engine.h");

Definition at line 165 of file movement.c.

**4.11.2.4 move\_p2()**

```
int move_p2 (
    player_t * players,
    unsigned int num_pawn,
    char * str,
    board_t * board )
```

Move a given number of pawn of the second player.

It is very similar to move\_p1 but it is different because the directions of player 2 move inversely with respect to player 1, for example if I have to go in the left direction of player 1, the direction of player 2 is not left, but bottom left. This is due to the fact that in the chessboard player 1 is in the last lines, instead player 2 in the first ones, so if I move to the left of player 1 the y coordinate of the pawn decreases, on the contrary in player 2 that when I have to go on the left the y coordinate increases (as in fact when compared to player 1 I go to the bottom left and then reusing the functions of player 1).

There are 4 string directions :

- > "l" which means left respect the num\_pawn pawn.
- > "r" which means right respect the num\_pawn pawn.
- > "botr" which means bottom right respect the num\_pawn pawn.
- > "botl" which means bottom left respect the num\_pawn pawn.

The 4 directions of player 2 respect player 1 :

- > "l" that respect player 1 it is "botl".
- > "r" that respect player 1 it is "botr".
- > "botr" that respect player 1 it is "r".
- > "l" that respect player 1 it is "l".

**Parameters**

<i>players</i>	Pointer to the players array.
<i>num_pawn</i>	Number of the pawn to move.
<i>str</i>	String of direction where to move the pawn.
<i>board</i>	Pointer to the board.

**Returns**

int Return a number  $\geq 0$  if the num\_pawn ate and matches to the number of enemy pawn that it was eaten, -1 if the num\_pawn move without eat, -2 if did nothing, -4 if there is an error in can\_eat function(#include "../game\_engine/game\_engine.h");

Definition at line 182 of file movement.c.

**4.12 src/user\_interaction/user\_interaction.c File Reference**

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include "../colors/colors.h"
#include "../game_engine/game_engine.h"
#include "../ia/ia.h"
#include "../memory_management/memory_management.h"
#include "../movement/movement.h"
#include "user_interaction.h"
```

**Functions**

- void [print\\_board](#) ([board\\_t](#) t, unsigned int cifre, unsigned nPl, char char\_p1, char char\_p2)  
*Print the board of game from the prospective of a given number of player.*
- void [print\\_player](#) ([player\\_t](#) \*players, unsigned int nPl)  
*Print the most important fields of a given number of player.*
- unsigned int [while\\_select\\_nPawn](#) ([player\\_t](#) \*players, unsigned int nPl)  
*Ask the player for a pawn number to select.*
- unsigned int [round\\_player](#) ([player\\_t](#) \*players, [board\\_t](#) \*t, unsigned int nPl)  
*Run the game round player.*
- unsigned int [round\\_choice](#) ()  
*Determine which player goes first.*
- int [game](#) (unsigned int gameMode)  
*Allocate memory for structs(with call of create function), use in-game memory in any mode, and finally free up memory(with call of destroy function).*
- int [checkInt](#) (int scanfValue)  
*Check if the inserted value is an integer.*
- void [menu](#) ()  
*Cleans the terminal and launches the main game menu.*

**4.12.1 Function Documentation****4.12.1.1 checkInt()**

```
int checkInt (
    int scanfValue )
```

Check if the inserted value is an integer.

**Parameters**

<i>scanfValue</i>	Value of return of the scanf.
-------------------	-------------------------------

**Returns**

int Return correct value.

Definition at line 624 of file user\_interaction.c.

**4.12.1.2 game()**

```
int game (
    unsigned int x )
```

Allocate memory for structs(with call of create function), use in-game memory in any mode, and finally free up memory(with call of destroy function).

**Parameters**

<i>x</i>	If 0 the game mode is player vs player ,else is player vs ia.
----------	---

**Returns**

int

Definition at line 359 of file user\_interaction.c.

**4.12.1.3 menu()**

```
void menu ( )
```

Cleans the terminal and launches the main game menu.

Definition at line 637 of file user\_interaction.c.

**4.12.1.4 print\_board()**

```
void print_board (
    board_t t,
    unsigned int cifre,
    unsigned npl,
    char char_p1,
    char char_p2 )
```

Print the board of game from the prospective of a given number of player.

## Parameters

<i>t</i>	Instance of the board.
<i>cifre</i>	Total number of space stored for the label of a pawn.
<i>npl</i>	Number of the player.
<i>char_p1</i>	Color of the first player.
<i>char_p2</i>	Color of the second player.

Definition at line 15 of file user\_interaction.c.

#### 4.12.1.5 print\_player()

```
void print_player (
    player_t * players,
    unsigned int nPl )
```

Print the most important fields of a given number of player.

## Parameters

<i>Players</i>	array wich contains the two players.
<i>nPl</i>	Number that identify the player.

Definition at line 162 of file user\_interaction.c.

#### 4.12.1.6 round\_choice()

```
unsigned int round_choice ( )
```

Determine which player goes first.

## Returns

unsigned int Return 0 if Player 1 goes first,1 if Player 2 goes first.

Definition at line 306 of file user\_interaction.c.

#### 4.12.1.7 round\_player()

```
unsigned int round_player (
    player_t * players,
    board_t * t,
    unsigned int nPl )
```

Run the game round player.

**Parameters**

<i>players</i>	Pointer to the players array.
<i>t</i>	Pointer to the board.
<i>nPl</i>	Number that identify the player.

**Returns**

unsigned int Return 4 for continue the while loop to play the game in function game.

Definition at line 216 of file user\_interaction.c.

**4.12.1.8 while\_select\_nPawn()**

```
unsigned int while_select_nPawn (
    player_t * players,
    unsigned int nPl )
```

Ask the player for a pawn number to select.

**Parameters**

<i>players</i>	Pointer to the players array.
<i>nPl</i>	Number that identify the player.

**Returns**

unsigned int Return a number of a pawn.

Definition at line 188 of file user\_interaction.c.

**4.13 src/user\_interaction/user\_interaction.h File Reference****Functions**

- void `print_board` (`board_t` t, unsigned int cifre, unsigned npl, char char\_p1, char char\_p2)  
*Print the board of game from the prospective of a given number of player.*
- void `print_player` (`player_t` \*players, unsigned int nPl)  
*Print the most important fields of a given number of player.*
- unsigned int `while_select_nPawn` (`player_t` \*players, unsigned int nPl)  
*Ask the player for a pawn number to select.*
- unsigned int `round_player` (`player_t` \*players, `board_t` \*t, unsigned int nPl)  
*Run the game round player.*
- unsigned int `round_choice` ()  
*Determine which player goes first.*



- int `game` (unsigned int x)  
*Allocate memory for structs(with call of create function), use in-game memory in any mode, and finally free up memory(with call of destroy function).*
- int `checkInt` (int scanfValue)  
*Check if the inserted value is an integer.*
- void `menu` ()  
*Cleans the terminal and launches the main game menu.*

### 4.13.1 Detailed Description

#### Author

Diego Passarella, Davide Pasqual, Michelle Ravagnan

#### Version

1.0.2

#### Date

2021-01-17

#### Copyright

Copyright (c) 2021

### 4.13.2 Function Documentation

#### 4.13.2.1 `checkInt()`

```
int checkInt (  
    int scanfValue )
```

Check if the inserted value is an integer.

#### Parameters

<code>scanfValue</code>	Value of return of the scanf.
-------------------------	-------------------------------

#### Returns

int Return correct value.

Definition at line 624 of file user\_interaction.c.

#### 4.13.2.2 game()

```
int game (
    unsigned int x )
```

Allocate memory for structs(with call of create function), use in-game memory in any mode, and finally free up memory(with call of destroy function).

##### Parameters

x	If 0 the game mode is player vs player ,else is player vs ia.
---	---

##### Returns

int

Definition at line 359 of file user\_interaction.c.

#### 4.13.2.3 menu()

```
void menu ( )
```

Cleans the terminal and launches the main game menu.

Definition at line 637 of file user\_interaction.c.

#### 4.13.2.4 print\_board()

```
void print_board (
    board_t t,
    unsigned int cifre,
    unsigned npl,
    char char_p1,
    char char_p2 )
```

Print the board of game from the prospective of a given number of player.

##### Parameters

t	Instance of the board.
cifre	Total number of space stored for the label of a pawn.
npl	Number of the player.
char_p1	Color of the first player.
char_p2	Color of the second player.

Definition at line 15 of file user\_interaction.c.

#### 4.13.2.5 print\_player()

```
void print_player (
    player_t * players,
    unsigned int nPl )
```

Print the most important fields of a given number of player.

##### Parameters

<i>Players</i>	array wich contains the two players.
<i>nPl</i>	Number that identify the player.

Definition at line 162 of file user\_interaction.c.

#### 4.13.2.6 round\_choice()

```
unsigned int round_choice ( )
```

Determine which player goes first.

##### Returns

unsigned int Return 0 if Player 1 goes first,1 if Player 2 goes first.

Definition at line 306 of file user\_interaction.c.

#### 4.13.2.7 round\_player()

```
unsigned int round_player (
    player_t * players,
    board_t * t,
    unsigned int nPl )
```

Run the game round player.

##### Parameters

<i>players</i>	Pointer to the players array.
<i>t</i>	Pointer to the board.
<i>nPl</i>	Number that identify the player.

**Returns**

unsigned int Return 4 for continue the while loop to play the game in function game.

Definition at line 216 of file user\_interaction.c.

**4.13.2.8 while\_select\_nPawn()**

```
unsigned int while_select_nPawn (  
    player_t * players,  
    unsigned int nPl )
```

Ask the player for a pawn number to select.

**Parameters**

<i>players</i>	Pointer to the players array.
<i>nPl</i>	Number that identify the player.

**Returns**

unsigned int Return a number of a pawn.

Definition at line 188 of file user\_interaction.c.

# Index

all\_blocked  
    game\_engine.c, [28](#)  
    game\_engine.h, [42](#)

BBLK  
    colors.h, [16](#)

BBLU  
    colors.h, [16](#)

BCYN  
    colors.h, [17](#)

BGRN  
    colors.h, [17](#)

BHBLK  
    colors.h, [17](#)

BHBLU  
    colors.h, [17](#)

BHCYN  
    colors.h, [17](#)

BHGRN  
    colors.h, [17](#)

BHMAG  
    colors.h, [18](#)

BHRED  
    colors.h, [18](#)

BHWHT  
    colors.h, [18](#)

BHYEL  
    colors.h, [18](#)

BLK  
    colors.h, [18](#)

BLKB  
    colors.h, [18](#)

BLKHB  
    colors.h, [19](#)

BLU  
    colors.h, [19](#)

BLUB  
    colors.h, [19](#)

BLUHB  
    colors.h, [19](#)

BMAG  
    colors.h, [19](#)

board, [5](#)  
    mat, [5](#)  
    n\_cols, [6](#)  
    n\_rows, [6](#)

board\_t  
    game\_engine.h, [53](#)

BRED  
    colors.h, [19](#)

BWHT  
    colors.h, [20](#)

BYEL  
    colors.h, [20](#)

call\_minimax  
    ia.c, [55](#)  
    ia.h, [62](#)

can\_eat  
    game\_engine.c, [29](#)  
    game\_engine.h, [42](#)

canMove  
    pawn, [7](#)

char\_converter  
    game\_engine.c, [29](#)  
    game\_engine.h, [43](#)

check\_canMove  
    game\_engine.c, [30](#)  
    game\_engine.h, [43](#)

check\_char\_color  
    game\_engine.c, [30](#)  
    game\_engine.h, [44](#)

check\_directions  
    game\_engine.c, [30](#)  
    game\_engine.h, [44](#)

check\_player  
    game\_engine.c, [31](#)  
    game\_engine.h, [45](#)

check\_spot  
    game\_engine.c, [31](#)  
    game\_engine.h, [45](#)

check\_string  
    game\_engine.c, [32](#)  
    game\_engine.h, [46](#)

check\_while  
    game\_engine.c, [32](#)  
    game\_engine.h, [46](#)

checkInt  
    user\_interaction.c, [83](#)  
    user\_interaction.h, [87](#)

cima  
    pawn, [7](#)

color  
    player, [9](#)

colors.c  
    printColor, [14](#)  
    printTextColor, [14](#)  
    setWhite, [14](#)

colors.h  
    BBLK, [16](#)

- BBLU, [16](#)
- BCYN, [17](#)
- BGRN, [17](#)
- BHBLK, [17](#)
- BHBLU, [17](#)
- BHCYN, [17](#)
- BHGRN, [17](#)
- BHMAG, [18](#)
- BHRED, [18](#)
- BHWHT, [18](#)
- BHYEL, [18](#)
- BLK, [18](#)
- BLKB, [18](#)
- BLKHB, [19](#)
- BLU, [19](#)
- BLUB, [19](#)
- BLUHB, [19](#)
- BMAG, [19](#)
- BRED, [19](#)
- BWHT, [20](#)
- BYEL, [20](#)
- CYN, [20](#)
- CYNB, [20](#)
- CYNHB, [20](#)
- GRN, [20](#)
- GRNB, [21](#)
- GRNHB, [21](#)
- HBLK, [21](#)
- HBLU, [21](#)
- HCYN, [21](#)
- HGRN, [21](#)
- HMAG, [22](#)
- HRED, [22](#)
- HWHT, [22](#)
- HYEL, [22](#)
- MAG, [22](#)
- MAGB, [22](#)
- MAGHB, [23](#)
- printColor, [26](#)
- printTextColor, [26](#)
- RED, [23](#)
- REDB, [23](#)
- REDHB, [23](#)
- reset, [23](#)
- setWhite, [26](#)
- UBLK, [23](#)
- UBLU, [24](#)
- UCYN, [24](#)
- UGRN, [24](#)
- UMAG, [24](#)
- URED, [24](#)
- UWHT, [24](#)
- UYEL, [25](#)
- WHT, [25](#)
- WHTB, [25](#)
- WHTHB, [25](#)
- YEL, [25](#)
- YELB, [25](#)
- YELHB, [26](#)
- coord
  - game\_engine.h, [41](#)
- coordinate
  - pawn, [7](#)
- copy\_board
  - memory\_management.c, [68](#)
  - memory\_management.h, [71](#)
- create\_board
  - memory\_management.c, [68](#)
  - memory\_management.h, [72](#)
- create\_pawns
  - memory\_management.c, [68](#)
  - memory\_management.h, [72](#)
- CYN
  - colors.h, [20](#)
- CYNB
  - colors.h, [20](#)
- CYNHB
  - colors.h, [20](#)
- destroy\_board
  - memory\_management.c, [69](#)
  - memory\_management.h, [73](#)
- destroy\_player
  - memory\_management.c, [69](#)
  - memory\_management.h, [73](#)
- destroy\_value\_minimax
  - ia.c, [56](#)
  - ia.h, [63](#)
- dim\_board
  - game\_engine.h, [42](#)
- dim\_label
  - pawn, [7](#)
- dim\_pawns
  - player, [9](#)
- directions
  - valueMinimax, [11](#)
- eat
  - movement.c, [75](#)
  - movement.h, [79](#)
- evaluate\_score
  - ia.c, [56](#)
  - ia.h, [63](#)
- flag
  - game\_engine.h, [42](#)
- game
  - user\_interaction.c, [84](#)
  - user\_interaction.h, [87](#)
- game\_engine.c
  - all\_blocked, [28](#)
  - can\_eat, [29](#)
  - char\_converter, [29](#)
  - check\_canMove, [30](#)
  - check\_char\_color, [30](#)
  - check\_directions, [30](#)

- check\_player, 31
- check\_spot, 31
- check\_string, 32
- check\_while, 32
- initialize\_board, 33
- int\_converter, 33
- is\_empty, 34
- is\_in, 34
- is\_notstuck, 34
- is\_selected, 35
- is\_victory, 35
- max\_pawns, 36
- must\_eat, 36
- pawn\_promotion, 37
- print\_directions, 37
- remove\_pawn, 37
- reset\_moves\_pawns, 38
- set\_moves\_pawn, 38
- update\_board, 39
- uppercase, 39
- game\_engine.h
  - all\_blocked, 42
  - board\_t, 53
  - can\_eat, 42
  - char\_converter, 43
  - check\_canMove, 43
  - check\_char\_color, 44
  - check\_directions, 44
  - check\_player, 45
  - check\_spot, 45
  - check\_string, 46
  - check\_while, 46
  - coord, 41
  - dim\_board, 42
  - flag, 42
  - initialize\_board, 47
  - int\_converter, 47
  - is\_empty, 47
  - is\_in, 48
  - is\_notstuck, 48
  - is\_selected, 49
  - is\_victory, 49
  - max\_pawns, 50
  - must\_eat, 50
  - pawn\_promotion, 50
  - pawn\_t, 53
  - player\_t, 54
  - point\_t, 54
  - print\_directions, 51
  - remove\_pawn, 51
  - reset\_moves\_pawns, 52
  - set\_moves\_pawn, 52
  - update\_board, 53
  - uppercase, 53
- grade
  - pawn, 7
- GRN
  - colors.h, 20
- GRNB
  - colors.h, 21
- GRNHB
  - colors.h, 21
- HBLK
  - colors.h, 21
- HBLU
  - colors.h, 21
- HCYN
  - colors.h, 21
- HGRN
  - colors.h, 21
- HMAG
  - colors.h, 22
- HRED
  - colors.h, 22
- HWHT
  - colors.h, 22
- HYEL
  - colors.h, 22
- ia.c
  - call\_minimax, 55
  - destroy\_value\_minimax, 56
  - evaluate\_score, 56
  - interrupt\_minimax, 56
  - last\_move, 58
  - max, 58
  - minimax, 59
  - print\_minimax, 59
  - round\_ia\_minimax, 60
  - round\_ia\_random, 60
- ia.h
  - call\_minimax, 62
  - destroy\_value\_minimax, 63
  - evaluate\_score, 63
  - interrupt\_minimax, 64
  - last\_move, 64
  - max, 64
  - minimax, 65
  - print\_minimax, 66
  - round\_ia\_minimax, 66
  - round\_ia\_random, 66
  - valueMinimax\_t, 62
- initialize\_board
  - game\_engine.c, 33
  - game\_engine.h, 47
- int\_converter
  - game\_engine.c, 33
  - game\_engine.h, 47
- interrupt\_minimax
  - ia.c, 56
  - ia.h, 64
- is\_empty
  - game\_engine.c, 34
  - game\_engine.h, 47
- is\_in
  - game\_engine.c, 34

- game\_engine.h, 48
- is\_notstuck
  - game\_engine.c, 34
  - game\_engine.h, 48
- is\_selected
  - game\_engine.c, 35
  - game\_engine.h, 49
- is\_victory
  - game\_engine.c, 35
  - game\_engine.h, 49
- isPromoted
  - pawn, 7
- label
  - pawn, 8
- last\_move
  - ia.c, 58
  - ia.h, 64
- MAG
  - colors.h, 22
- MAGB
  - colors.h, 22
- MAGHB
  - colors.h, 23
- main
  - main.c, 13
- main.c, 13
  - main, 13
- mat
  - board, 5
- max
  - ia.c, 58
  - ia.h, 64
- max\_pawns
  - game\_engine.c, 36
  - game\_engine.h, 50
- memory\_management.c
  - copy\_board, 68
  - create\_board, 68
  - create\_pawns, 68
  - destroy\_board, 69
  - destroy\_player, 69
  - player\_copy, 70
  - restore\_copy, 70
- memory\_management.h
  - copy\_board, 71
  - create\_board, 72
  - create\_pawns, 72
  - destroy\_board, 73
  - destroy\_player, 73
  - player\_copy, 73
  - restore\_copy, 74
- menu
  - user\_interaction.c, 84
  - user\_interaction.h, 88
- minimax
  - ia.c, 59
  - ia.h, 65
- move\_noeat
  - movement.c, 76
  - movement.h, 80
- move\_p1
  - movement.c, 77
  - movement.h, 81
- move\_p2
  - movement.c, 78
  - movement.h, 82
- movement.c
  - eat, 75
  - move\_noeat, 76
  - move\_p1, 77
  - move\_p2, 78
- movement.h
  - eat, 79
  - move\_noeat, 80
  - move\_p1, 81
  - move\_p2, 82
- must\_eat
  - game\_engine.c, 36
  - game\_engine.h, 50
- n\_cols
  - board, 6
- n\_rows
  - board, 6
- pawn, 6
  - canMove, 7
  - cima, 7
  - coordinate, 7
  - dim\_label, 7
  - grade, 7
  - isPromoted, 7
  - label, 8
- pawn\_promotion
  - game\_engine.c, 37
  - game\_engine.h, 50
- pawn\_t
  - game\_engine.h, 53
- pawns
  - player, 9
- player, 8
  - color, 9
  - dim\_pawns, 9
  - pawns, 9
- player\_copy
  - memory\_management.c, 70
  - memory\_management.h, 73
- player\_t
  - game\_engine.h, 54
- point, 9
  - x, 10
  - y, 10
- point\_t
  - game\_engine.h, 54
- print\_board
  - user\_interaction.c, 84



- user\_interaction.h, 88
- print\_directions
  - game\_engine.c, 37
  - game\_engine.h, 51
- print\_minimax
  - ia.c, 59
  - ia.h, 66
- print\_player
  - user\_interaction.c, 85
  - user\_interaction.h, 89
- printColor
  - colors.c, 14
  - colors.h, 26
- printTextColor
  - colors.c, 14
  - colors.h, 26
- RED
  - colors.h, 23
- REDB
  - colors.h, 23
- REDHB
  - colors.h, 23
- remove\_pawn
  - game\_engine.c, 37
  - game\_engine.h, 51
- reset
  - colors.h, 23
- reset\_moves\_pawns
  - game\_engine.c, 38
  - game\_engine.h, 52
- restore\_copy
  - memory\_management.c, 70
  - memory\_management.h, 74
- round\_choice
  - user\_interaction.c, 85
  - user\_interaction.h, 89
- round\_ia\_minimax
  - ia.c, 60
  - ia.h, 66
- round\_ia\_random
  - ia.c, 60
  - ia.h, 66
- round\_player
  - user\_interaction.c, 85
  - user\_interaction.h, 89
- set\_moves\_pawn
  - game\_engine.c, 38
  - game\_engine.h, 52
- setWhite
  - colors.c, 14
  - colors.h, 26
- src/colors/colors.c, 13
- src/colors/colors.h, 15
- src/game\_engine/game\_engine.c, 27
- src/game\_engine/game\_engine.h, 39
- src/ia/ia.c, 54
- src/ia/ia.h, 61

- src/memory\_management/memory\_management.c, 67
- src/memory\_management/memory\_management.h, 71
- src/movement/movement.c, 74
- src/movement/movement.h, 79
- src/user\_interaction/user\_interaction.c, 83
- src/user\_interaction/user\_interaction.h, 86
- UBLK
  - colors.h, 23
- UBLU
  - colors.h, 24
- UCYN
  - colors.h, 24
- UGRN
  - colors.h, 24
- UMAG
  - colors.h, 24
- update\_board
  - game\_engine.c, 39
  - game\_engine.h, 53
- uppercase
  - game\_engine.c, 39
  - game\_engine.h, 53
- URED
  - colors.h, 24
- user\_interaction.c
  - checkInt, 83
  - game, 84
  - menu, 84
  - print\_board, 84
  - print\_player, 85
  - round\_choice, 85
  - round\_player, 85
  - while\_select\_nPawn, 86
- user\_interaction.h
  - checkInt, 87
  - game, 87
  - menu, 88
  - print\_board, 88
  - print\_player, 89
  - round\_choice, 89
  - round\_player, 89
  - while\_select\_nPawn, 90
- UWHT
  - colors.h, 24
- UYEL
  - colors.h, 25
- value
  - valueMinimax, 11
- valueMinimax, 10
  - directions, 11
  - value, 11
- valueMinimax\_t
  - ia.h, 62
- while\_select\_nPawn
  - user\_interaction.c, 86
  - user\_interaction.h, 90

## WHT

colors.h, [25](#)

## WHTB

colors.h, [25](#)

## WHTHB

colors.h, [25](#)

## x

point, [10](#)

## y

point, [10](#)

## YEL

colors.h, [25](#)

## YELB

colors.h, [25](#)

## YELHB

colors.h, [26](#)