

## Othello Finale

3.0

Generated by Doxygen 1.8.6

Mon Apr 27 2015 07:09:21



# Contents

<b>1</b>	<b>Othello</b>	<b>1</b>
<b>2</b>	<b>Data Structure Index</b>	<b>3</b>
2.1	Data Structures . . . . .	3
<b>3</b>	<b>File Index</b>	<b>5</b>
3.1	File List . . . . .	5
<b>4</b>	<b>Data Structure Documentation</b>	<b>7</b>
4.1	Game Struct Reference . . . . .	7
4.1.1	Detailed Description . . . . .	7
4.1.2	Field Documentation . . . . .	7
4.1.2.1	boardsCursor . . . . .	7
4.1.2.2	coefs . . . . .	7
4.1.2.3	cursor . . . . .	7
4.1.2.4	moves . . . . .	8
4.1.2.5	movesCursor . . . . .	8
4.1.2.6	oth . . . . .	8
4.1.2.7	oths . . . . .	8
4.1.2.8	p1 . . . . .	8
4.1.2.9	p2 . . . . .	8
4.1.2.10	saved . . . . .	8
4.1.2.11	turn . . . . .	8
4.2	Genome Struct Reference . . . . .	8
4.2.1	Detailed Description . . . . .	8
4.2.2	Field Documentation . . . . .	9
4.2.2.1	coefs . . . . .	9
4.2.2.2	fitness . . . . .	9
4.3	Move Struct Reference . . . . .	9
4.3.1	Detailed Description . . . . .	9
4.3.2	Field Documentation . . . . .	9
4.3.2.1	color . . . . .	9

4.3.2.2	flip	9
4.3.2.3	position	9
4.4	Othellier Struct Reference	9
4.4.1	Detailed Description	10
4.4.2	Field Documentation	10
4.4.2.1	array	10
4.4.2.2	piecesNb	10
4.4.2.3	turn	10
4.5	Player Struct Reference	10
4.5.1	Detailed Description	10
4.5.2	Field Documentation	10
4.5.2.1	player_name	10
4.5.2.2	player_type	11
4.6	Population Struct Reference	11
4.6.1	Detailed Description	11
4.6.2	Field Documentation	11
4.6.2.1	genNb	11
4.6.2.2	genomes	11
<b>5</b>	<b>File Documentation</b>	<b>13</b>
5.1	ai.c File Reference	13
5.1.1	Detailed Description	13
5.1.2	Function Documentation	13
5.1.2.1	evaluate_corner_square_pattern	13
5.1.2.2	evaluate_long_alignments	14
5.1.2.3	evaluation_function	14
5.1.2.4	find_legal_moves	14
5.1.2.5	min_max	15
5.1.2.6	number_of_corners	16
5.1.2.7	positions_strength	16
5.2	ai.h File Reference	17
5.2.1	Detailed Description	17
5.2.2	Function Documentation	17
5.2.2.1	evaluate_corner_square_pattern	17
5.2.2.2	evaluate_long_alignments	18
5.2.2.3	evaluation_function	18
5.2.2.4	find_legal_moves	18
5.2.2.5	min_max	19
5.2.2.6	number_of_corners	20
5.2.2.7	positions_strength	20

5.3	constants.h File Reference	20
5.3.1	Detailed Description	23
5.3.2	Macro Definition Documentation	23
5.3.2.1	GEN_ALGO	23
5.4	events.c File Reference	23
5.4.1	Detailed Description	24
5.4.2	Function Documentation	24
5.4.2.1	event_cancel	24
5.4.2.2	event_help	24
5.4.2.3	event_load	24
5.4.2.4	event_new	25
5.4.2.5	event_other	25
5.4.2.6	event_quit	25
5.4.2.7	event_save	26
5.4.2.8	event_uncancel	27
5.5	events.h File Reference	27
5.5.1	Detailed Description	28
5.5.2	Function Documentation	28
5.5.2.1	event_cancel	28
5.5.2.2	event_help	28
5.5.2.3	event_load	28
5.5.2.4	event_new	28
5.5.2.5	event_other	28
5.5.2.6	event_quit	29
5.5.2.7	event_save	29
5.5.2.8	event_uncancel	29
5.6	game.c File Reference	29
5.6.1	Detailed Description	30
5.6.2	Function Documentation	30
5.6.2.1	destroy_end_moves_tab	30
5.6.2.2	destroy_game	30
5.6.2.3	end_of_game	30
5.6.2.4	end_of_game_action	31
5.6.2.5	has_legal_move	31
5.6.2.6	init_game	31
5.6.2.7	is_computer_game	31
5.6.2.8	load_coefs	32
5.6.2.9	play_ai	32
5.6.2.10	update_turn	32
5.6.2.11	whose_turn_is_it	32

5.7	game.h File Reference	33
5.7.1	Detailed Description	33
5.7.2	Function Documentation	33
5.7.2.1	destroy_end_moves_tab	33
5.7.2.2	destroy_game	34
5.7.2.3	end_of_game	34
5.7.2.4	end_of_game_action	34
5.7.2.5	has_legal_move	34
5.7.2.6	init_game	35
5.7.2.7	is_computer_game	35
5.7.2.8	load_coefs	35
5.7.2.9	play_ai	35
5.7.2.10	update_turn	36
5.7.2.11	whose_turn_is_it	37
5.8	genetic.c File Reference	37
5.8.1	Detailed Description	38
5.8.2	Function Documentation	38
5.8.2.1	destroy_population	38
5.8.2.2	fitness	38
5.8.2.3	genetic_algorithm	38
5.8.2.4	initial_population	38
5.8.2.5	mutation	38
5.8.2.6	next_generation	39
5.8.2.7	random_tenth	39
5.8.2.8	reproduction	39
5.8.2.9	save_coefs	39
5.8.2.10	search_best_genome	40
5.9	genetic.h File Reference	40
5.9.1	Detailed Description	41
5.9.2	Function Documentation	41
5.9.2.1	destroy_population	41
5.9.2.2	fitness	41
5.9.2.3	genetic_algorithm	41
5.9.2.4	initial_population	41
5.9.2.5	mutation	41
5.9.2.6	next_generation	42
5.9.2.7	random_tenth	42
5.9.2.8	reproduction	42
5.9.2.9	save_coefs	42
5.9.2.10	search_best_genome	43

5.10 graphics.c File Reference . . . . .	43
5.10.1 Detailed Description . . . . .	43
5.11 graphics.h File Reference . . . . .	43
5.11.1 Detailed Description . . . . .	44
5.12 io.c File Reference . . . . .	44
5.12.1 Detailed Description . . . . .	45
5.12.2 Function Documentation . . . . .	45
5.12.2.1 are_good_coordinates . . . . .	45
5.12.2.2 ask_YN . . . . .	45
5.12.2.3 char2int . . . . .	46
5.12.2.4 contains_only_letters . . . . .	46
5.12.2.5 display_othellier . . . . .	46
5.12.2.6 empty_buffer . . . . .	46
5.12.2.7 exists_file . . . . .	47
5.12.2.8 game_saving . . . . .	48
5.12.2.9 int2str . . . . .	48
5.12.2.10 is_a_letter . . . . .	48
5.12.2.11 load_game . . . . .	49
5.12.2.12 str2upper . . . . .	50
5.13 io.h File Reference . . . . .	50
5.13.1 Detailed Description . . . . .	51
5.13.2 Function Documentation . . . . .	51
5.13.2.1 are_good_coordinates . . . . .	51
5.13.2.2 ask_YN . . . . .	51
5.13.2.3 char2int . . . . .	51
5.13.2.4 contains_only_letters . . . . .	52
5.13.2.5 display_othellier . . . . .	52
5.13.2.6 empty_buffer . . . . .	52
5.13.2.7 exists_file . . . . .	52
5.13.2.8 game_saving . . . . .	53
5.13.2.9 int2str . . . . .	54
5.13.2.10 is_a_letter . . . . .	54
5.13.2.11 load_game . . . . .	54
5.13.2.12 str2upper . . . . .	55
5.14 main.c File Reference . . . . .	55
5.14.1 Detailed Description . . . . .	55
5.14.2 Function Documentation . . . . .	55
5.14.2.1 main . . . . .	55
5.15 main.h File Reference . . . . .	55
5.15.1 Detailed Description . . . . .	56

5.15.2	Function Documentation	56
5.15.2.1	main	56
5.16	move.c File Reference	56
5.16.1	Detailed Description	57
5.16.2	Function Documentation	57
5.16.2.1	cancel_last_move	57
5.16.2.2	create_move	57
5.16.2.3	move_processing	57
5.16.2.4	play_move	58
5.16.2.5	uncancel_move	58
5.17	move.h File Reference	58
5.17.1	Detailed Description	59
5.17.2	Function Documentation	59
5.17.2.1	cancel_last_move	59
5.17.2.2	create_move	59
5.17.2.3	move_processing	59
5.17.2.4	play_move	60
5.17.2.5	uncancel_move	60
5.18	othellier.c File Reference	60
5.18.1	Detailed Description	61
5.18.2	Function Documentation	61
5.18.2.1	destroy_end_boards_tab	61
5.18.2.2	game_board_copy	61
5.18.2.3	has_alignment	61
5.18.2.4	init_othellier	62
5.18.2.5	is_legal	62
5.18.2.6	update_flip	62
5.18.2.7	update_othellier	63
5.19	othellier.h File Reference	64
5.19.1	Detailed Description	64
5.19.2	Function Documentation	64
5.19.2.1	destroy_end_boards_tab	64
5.19.2.2	game_board_copy	65
5.19.2.3	has_alignment	65
5.19.2.4	init_othellier	65
5.19.2.5	is_legal	65
5.19.2.6	update_flip	66
5.19.2.7	update_othellier	66
5.20	structures.h File Reference	66
5.20.1	Detailed Description	67



5.21 user.c File Reference . . . . .	67
5.21.1 Detailed Description . . . . .	68
5.21.2 Function Documentation . . . . .	68
5.21.2.1 ask_load_game . . . . .	68
5.21.2.2 ask_new_game . . . . .	68
5.21.2.3 ask_save_game . . . . .	68
5.21.2.4 choose_game_type . . . . .	68
5.21.2.5 choose_player_color . . . . .	69
5.21.2.6 display_help . . . . .	69
5.21.2.7 display_usage . . . . .	69
5.21.2.8 enter_one_name . . . . .	69
5.21.2.9 enter_players_names . . . . .	69
5.21.2.10 give_computer_name . . . . .	70
5.22 user.h File Reference . . . . .	70
5.22.1 Detailed Description . . . . .	71
5.22.2 Function Documentation . . . . .	71
5.22.2.1 ask_load_game . . . . .	71
5.22.2.2 ask_new_game . . . . .	71
5.22.2.3 ask_save_game . . . . .	71
5.22.2.4 choose_game_type . . . . .	71
5.22.2.5 choose_player_color . . . . .	72
5.22.2.6 display_help . . . . .	72
5.22.2.7 display_usage . . . . .	72
5.22.2.8 enter_one_name . . . . .	72
5.22.2.9 enter_players_names . . . . .	72
5.22.2.10 give_computer_name . . . . .	73
<b>Index</b>	<b>74</b>



# Chapter 1

## Othello

This is the documentation of the program Othello, developed by S. Guiol.

### RULES :

The Othello game is constituted of a board of 64 squares (8x8). The game is played with 2 players, the Black and the White.

At the beginning, each player has 2 pawns of his color in diagonal on the center of the board.  
The black player always goes first.

To make a move, the player has to put a pawn of his color on a square so that it makes the pawn(s) of his color. More specifically, he can put a pawn on a square only if, in at least one of the eight directions, two of his pawns form an alignment with an empty square. To form an alignment, his two pawns can not be separated by empty cases, unless they make an alignment in the direction of the empty square.

For each alignment formed, all the opponent's pawns located between two of his pawns (including the one he just moved) are captured.

If one player don't have any possible move, the turn passes to the other player.

The game stops when all the board is full or when the two players don't have any remaining move to make.

The winner is, at the end of the game, the one who has the more pawns of his color. If the two players have the same number of pawns, the game is a draw.

### USING THE SOFTWARE :

The software is easy to use. After launching, a little usage is displayed. If you need more help, type "H". The following commands are implemented :

N or NEW : Begins a new game

L or LOAD : Loads a game from a text file

S or SAVE : Saves the game in progress in a text file

H or HELP : Displays the help

C or CANCEL : Cancels the last move played

U or UNCANCEL : Uncancels the last canceled move

Q or QUIT : Exits the game

Typing in uppercase or lowercase doesn't matter.

While in a game, all commands above are recognised, unless you are performing a typing action (answering to a question). If you only want to play, when the board is displayed on the screen, just type the coordinates you want to move a pawn to.



## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

Game	Game structure . . . . .	7
Genome	Genome structure . . . . .	8
Move	Move structure . . . . .	9
Othellier	Othellier (game board) structure . . . . .	9
Player	Player structure . . . . .	10
Population	Population structure . . . . .	11



## Chapter 3

# File Index

### 3.1 File List

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

<a href="#">ai.c</a>	13
<a href="#">ai.h</a>	17
<a href="#">constants.h</a>	20
<a href="#">events.c</a>	23
<a href="#">events.h</a>	27
<a href="#">game.c</a>	29
<a href="#">game.h</a>	33
<a href="#">genetic.c</a>	37
<a href="#">genetic.h</a>	40
<a href="#">graphics.c</a>	43
<a href="#">graphics.h</a>	43
<a href="#">io.c</a>	44
<a href="#">io.h</a>	50
<a href="#">main.c</a>	55
<a href="#">main.h</a>	55
<a href="#">move.c</a>	56
<a href="#">move.h</a>	58
<a href="#">othellier.c</a>	60
<a href="#">othellier.h</a>	64
<a href="#">structures.h</a>	66
<a href="#">user.c</a>	67
<a href="#">user.h</a>	70





## Chapter 4

# Data Structure Documentation

### 4.1 Game Struct Reference

`Game` structure.

```
#include <structures.h>
```

#### Data Fields

- short `cursor`
- short `movesCursor`
- short `boardsCursor`
- short `turn`
- short `saved`
- `Player` `p1`
- `Player` `p2`
- `Move` `** moves`
- `Othellier` `** oths`
- `Othellier` `* oth`
- float `coefs` [`COEF_NB`]

#### 4.1.1 Detailed Description

`Game` structure.

#### 4.1.2 Field Documentation

##### 4.1.2.1 short `boardsCursor`

Cursor for the last cell of the boards array

##### 4.1.2.2 float `coefs`[`COEF_NB`]

Coefficients used by the evaluation function

##### 4.1.2.3 short `cursor`

Number of the move

#### 4.1.2.4 **Move\*\*** moves

Moves array (for saving)

#### 4.1.2.5 **short** movesCursor

Cursor for the last cell of the moves array

#### 4.1.2.6 **Othellier\*** oth

[Othellier](#) (game board)

#### 4.1.2.7 **Othellier\*\*** oths

[Game](#) boards array (for saving)

#### 4.1.2.8 **Player** p1

[Player](#) 1

#### 4.1.2.9 **Player** p2

[Player](#) 2

#### 4.1.2.10 **short** saved

Is the game saved ? 1 for yes, 0 for no

#### 4.1.2.11 **short** turn

Number of the player who has to play

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

- [structures.h](#)

## 4.2 Genome Struct Reference

[Genome](#) structure.

```
#include <structures.h>
```

### Data Fields

- float [coefs](#) [[COEF\\_NB](#)]
- short [fitness](#)

#### 4.2.1 Detailed Description

[Genome](#) structure.

## 4.2.2 Field Documentation

### 4.2.2.1 float coefs[COEF\_NB]

Coefficients array

### 4.2.2.2 short fitness

Fitness of the genome

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

- [structures.h](#)

## 4.3 Move Struct Reference

[Move](#) structure.

```
#include <structures.h>
```

### Data Fields

- short [position](#)
- short [color](#)
- short [flip](#) [MAX\_FLIP]

### 4.3.1 Detailed Description

[Move](#) structure.

## 4.3.2 Field Documentation

### 4.3.2.1 short color

Color of the player who makes the move

### 4.3.2.2 short flip[MAX\_FLIP]

Array for the existence of flip if the move is played

### 4.3.2.3 short position

Position of the move on the game board (array)

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

- [structures.h](#)

## 4.4 Othellier Struct Reference

[Othellier](#) (game board) structure.

```
#include <structures.h>
```

## Data Fields

- short [array](#) [[MAX\\_CASE](#)]
- short [piecesNb](#) [[PLAYER\\_NB](#)]
- short [turn](#)

### 4.4.1 Detailed Description

[Othellier](#) (game board) structure.

short : Short type has been preferred as int type due to memory use. The unsigned short type takes the same amount of memory as the short type.

### 4.4.2 Field Documentation

#### 4.4.2.1 short array[[MAX\\_CASE](#)]

Array for the game board

#### 4.4.2.2 short piecesNb[[PLAYER\\_NB](#)]

Array for the storage of the number of pawn for each player

#### 4.4.2.3 short turn

Number of the player who has the turn

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

- [structures.h](#)

## 4.5 Player Struct Reference

[Player](#) structure.

```
#include <structures.h>
```

## Data Fields

- char [player\\_name](#) [[MAX\\_NAME](#)]
- short [player\\_type](#)

### 4.5.1 Detailed Description

[Player](#) structure.

### 4.5.2 Field Documentation

#### 4.5.2.1 char player\_name[[MAX\\_NAME](#)]

String for the storage of the name of one player

#### 4.5.2.2 short player\_type

Type of the player (human/computer)

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

- [structures.h](#)

## 4.6 Population Struct Reference

[Population](#) structure.

```
#include <structures.h>
```

### Data Fields

- [Genome](#) \* [genomes](#)
- short [genNb](#)

#### 4.6.1 Detailed Description

[Population](#) structure.

#### 4.6.2 Field Documentation

##### 4.6.2.1 short genNb

Number of genomes in the population

##### 4.6.2.2 [Genome](#)\* [genomes](#)

Genomes array

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

- [structures.h](#)



## Chapter 5

# File Documentation

### 5.1 ai.c File Reference

```
#include "ai.h"
```

#### Functions

- short [number\\_of\\_corners](#) ([Othellier](#) \*oth, short [color](#))  
*This function counts the number of corner occupied by the player having the "color" color.*
- short [find\\_legal\\_moves](#) ([Game](#) \*game, short tab[])  
*This function finds all of the legal moves of the player who has the turn, and stores them in the array 'tab'.*
- short [evaluate\\_long\\_alignments](#) ([Othellier](#) \*oth, short [color](#), short position, short direction, short weight)  
*This function returns a value as important as the alignment is long.*
- short [evaluate\\_corner\\_square\\_pattern](#) ([Othellier](#) \*oth, short [color](#), short position, short strength)  
*This function returns a value depending on the square pattern size found from the corner passed in argument.*
- short [positions\\_strength](#) ([Game](#) \*game, short [color](#))  
*This function makes a sober evaluation of the game board for the color 'color'.*
- short [evaluation\\_function](#) ([Game](#) \*game, short [color](#), float coefs[])  
*This function makes a more exhaustive evaluation of the win possibilities of a player.*
- short [min\\_max](#) (short maximize, [Game](#) \*game, short playerAi, short depth, short height, short alpha, short beta, float coefs[])  
*This function applies the minimax algorithm on the game, using alpha-beta pruning.*

#### 5.1.1 Detailed Description

The [ai.c](#) file in which you can find all algorithms for the development of the Artificial Intelligence of the game

#### 5.1.2 Function Documentation

##### 5.1.2.1 short [evaluate\\_corner\\_square\\_pattern](#) ( [Othellier](#) \* [oth](#), short [color](#), short [position](#), short [strength](#) )

This function returns a value depending on the square pattern size found from the corner passed in argument.

**Parameters**

<i>oth</i>	: a pointer to an <a href="#">Othellier</a> structure
<i>color</i>	: the color of the player who may have the pattern
<i>position</i>	: the position of the corner we want to analyze
<i>strength</i>	: the strength to apply for a size 1 pattern

Gives a weight for the pattern considered. The bigger the pattern, the higher the weight.

#### 5.1.2.2 short evaluate\_long\_alignments ( [Othellier](#) \* *oth*, short *color*, short *position*, short *direction*, short *weight* )

This function returns a value as important as the alignment is long.

**Parameters**

<i>oth</i>	: a pointer to an <a href="#">Othellier</a> structure
<i>color</i>	: the color we want to examine
<i>position</i>	: the position from which we want to find the alignment
<i>direction</i>	: the direction of the alignment
<i>weight</i>	: the weight of each position of the alignment

**Returns**

the weight of the alignment

Gives a weight for the alignment considered. The longer the alignment is, the higher the weight is.

#### 5.1.2.3 short evaluation\_function ( [Game](#) \* *game*, short *color*, float *coefs*[] )

This function makes a more exhaustive evaluation of the win possibilities of a player.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color we want to examine
<i>coefs</i>	: a coefficients array

**Returns**

an evaluation of the game state

Applies coefficients on 4 different types of data : material, positions, mobility and the square pattern. Calls [positions\\_strength\(\)](#) for an evaluation of the positions of the pawns.

#### 5.1.2.4 short find\_legal\_moves ( [Game](#) \* *game*, short *tab*[] )

This function finds all of the legal moves of the player who has the turn, and stores them in the array 'tab[]'.

**Parameters**

<i>game</i>	: a pointer to <a href="#">Game</a> structure
<i>tab</i>	: the array in which we want to store the legal moves

**Returns**

the number of legal moves (the length of the array)

Updates the game board and tells, for each square, if the move is legal or not. If so, stores the position of the square in the array.



5.1.2.5 short min\_max ( short *maximize*, **Game** \* *game*, short *playerAi*, short *depth*, short *height*, short *alpha*, short *beta*, float *coefs*[] )

This function applies the minimax algorithm on the game, using alpha-beta pruning.

**Parameters**

<i>maximize</i>	: says if the algorithm must be in maximize position or in minimize
<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>playerAi</i>	: the color of the player who first called minimax (the AI player)
<i>depth</i>	: the depth the algorithm has to scour
<i>height</i>	: the depth min-max will scour, but height won't change during the algorithm. Used to verify if the current depth is the higher one
<i>alpha</i>	: the value alpha for alpha-beta pruning
<i>beta</i>	: the value beta for alpha-beta pruning
<i>coefs</i>	: the coefficients to use in the evaluation function

**Returns**

the evaluation if the algorithm is ot finished, the best position to play otherwise

Applies the famous minimax algorithm on the Othello game. Takes into cosideration the player who is the 'AI' and the fact that the turn may not be always to the opponent after a move is played. Alpha-Beta pruning has been implemented too.

**5.1.2.6 short number\_of\_corners ( Othellier \* oth, short color )**

This function counts the number of corner occupied by the player having the "color" color.

**Parameters**

<i>oth</i>	: a pointer to an <a href="#">Othellier</a> structure
<i>color</i>	: the color of the player we want to find the number of corners

**Returns**

the number of corners occupied

Tests, for each corner, if it is occupied by a pawn of the color "color". Increments the counter if so and returns the counter.

**5.1.2.7 short positions\_strength ( Game \* game, short color )**

This function makes a sober evaluation of the game board for the color 'color'.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color we want to examine

**Returns**

an evaluation of the board

Gives arbitrary values for most importants squarres of the board, as corners and borders.

**5.2 ai.h File Reference**

```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#include <time.h>
#include "constants.h"
#include "graphics.h"
#include "move.h"
#include "othellier.h"
#include "structures.h"
#include "user.h"
```

**Functions**

- short [number\\_of\\_corners](#) ([Othellier](#) \*oth, short [color](#))  
*This function counts the number of corner occupied by the player having the "color" color.*
- short [find\\_legal\\_moves](#) ([Game](#) \*game, short tab[])  
*This function finds all of the legal moves of the player who has the turn, and stores them in the array 'tab[]'.*
- short [evaluate\\_long\\_alignments](#) ([Othellier](#) \*oth, short [color](#), short position, short direction, short weight)  
*This function returns a value as important as the alignment is long.*
- short [evaluate\\_corner\\_square\\_pattern](#) ([Othellier](#) \*oth, short [color](#), short position, short strength)  
*This function returns a value depending on the squarre pattern size found from the corner passed in argument.*
- short [positions\\_strength](#) ([Game](#) \*game, short [color](#))  
*This function makes a sober evaluation of the game board for the color 'color'.*
- short [evaluation\\_function](#) ([Game](#) \*game, short [color](#), float coefs[])  
*This function makes a more exhaustive evaluation of the win possibilities of a player.*
- short [min\\_max](#) (short maximize, [Game](#) \*game, short playerAi, short depth, short height, short alpha, short beta, float coefs[])  
*This function applies the minimax algorithm on the game, using alpha-beta pruning.*

**5.2.1 Detailed Description**

The header file of the .c file [ai.c](#). Here you can find the prototypes of all functions of this file

**5.2.2 Function Documentation****5.2.2.1 short evaluate\_corner\_square\_pattern ( [Othellier](#) \* oth, short [color](#), short [position](#), short [strength](#) )**

This function returns a value depending on the squarre pattern size found from the corner passed in argument.

**Parameters**

<i>oth</i>	: a pointer to an <a href="#">Othellier</a> structure
<i>color</i>	: the color of the player who may have the pattern
<i>position</i>	: the position of the corner we want to analyze
<i>strength</i>	: the strength to apply for a size 1 pattern

Gives a weight for the pattern considered. The bigger the pattern, the higher the weight.

#### 5.2.2.2 short evaluate\_long\_alignments ( [Othellier](#) \* *oth*, short *color*, short *position*, short *direction*, short *weight* )

This function returns a value as important as the alignment is long.

**Parameters**

<i>oth</i>	: a pointer to an <a href="#">Othellier</a> structure
<i>color</i>	: the color we want to examine
<i>position</i>	: the position from which we want to find the alignment
<i>direction</i>	: the direction of the alignment
<i>weight</i>	: the weight of each position of the alignment

**Returns**

the weight of the alignment

Gives a weight for the alignment considered. The longer the alignment is, the higher the weight is.

#### 5.2.2.3 short evaluation\_function ( [Game](#) \* *game*, short *color*, float *coefs*[] )

This function makes a more exhaustive evaluation of the win possibilities of a player.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color we want to examine
<i>coefs</i>	: a coefficients array

**Returns**

an evaluation of the game state

Applies coefficients on 4 different types of data : material, positions, mobility and the square pattern. Calls [positions\\_strength\(\)](#) for an evaluation of the positions of the pawns.

#### 5.2.2.4 short find\_legal\_moves ( [Game](#) \* *game*, short *tab*[] )

This function finds all of the legal moves of the player who has the turn, and stores them in the array 'tab[]'.

**Parameters**

<i>game</i>	: a pointer to <a href="#">Game</a> structure
<i>tab</i>	: the array in which we want to store the legal moves

**Returns**

the number of legal moves (the length of the array)

Updates the game board and tells, for each square, if the move is legal or not. If so, stores the position of the square in the array.

5.2.2.5 short min\_max ( short *maximize*, **Game** \* *game*, short *playerAi*, short *depth*, short *height*, short *alpha*, short *beta*, float *coefs*[] )

This function applies the minimax algorithm on the game, using alpha-beta pruning.

**Parameters**

<i>maximize</i>	: says if the algorithm must be in maximize position or in minimize
<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>playerAi</i>	: the color of the player who first called minimax (the AI player)
<i>depth</i>	: the depth the algorithm has to scour
<i>height</i>	: the depth min-max will scour, but height won't change during the algorithm. Used to verify if the current depth is the higher one
<i>alpha</i>	: the value alpha for alpha-beta pruning
<i>beta</i>	: the value beta for alpha-beta pruning
<i>coefs</i>	: the coefficients to use in the evaluation function

**Returns**

the evaluation if the algorithm is ot finished, the best position to play otherwise

Applies the famous minimax algorithm on the Othello game. Takes into cosideration the player who is the 'AI' and the fact that the turn may not be always to the opponent after a move is played. Alpha-Beta pruning has been implemented too.

**5.2.2.6 short number\_of\_corners ( [Othellier](#) \* *oth*, short *color* )**

This function counts the number of corner occupied by the player having the "color" color.

**Parameters**

<i>oth</i>	: a pointer to an <a href="#">Othellier</a> structure
<i>color</i>	: the color of the player we want to find the number of corners

**Returns**

the number of corners occupied

Tests, for each corner, if it is occupied by a pawn of the color "color". Increments the counter if so and returns the counter.

**5.2.2.7 short positions\_strength ( [Game](#) \* *game*, short *color* )**

This function makes a sober evaluation of the game board for the color 'color'.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color we want to examine

**Returns**

an evaluation of the board

Gives arbitrary values for most importants squarres of the board, as corners and borders.

**5.3 constants.h File Reference****Macros**

- `#define YES 1`

- Macro Yes.
- #define NO 0
- Macro No.
- #define GEN\_ALGO 1
- Defines if the genetic algorithm is running or not.
- #define GLOBAL\_DEPTH 1
- The depth used in the min-max algorithm.
- #define GENERATION\_NB 35
- The number of generations.
- #define GAME\_NB 10
- The number of game to play between genomes.
- #define COEF\_NB 4
- The number of coefficients in the evaluation function.
- #define GENOMES\_NB 24
- The number of genomes (total initial population)
- #define MUT\_FACTOR 0.05
- Defines the probability of a gene mutation.
- #define MAX\_COEF 2.5
- The upper bound of the random number for each coefficient used in the genetic algorithm.
- #define COEFS\_FILENAME "GENETIC\_ALGORITHM\_DO\_NOT\_DELETE.txt"
- The name of the file in which the coefficients determined by the genetic algorithm are stored.
- #define ZERO 0
- Macro 0.
- #define MAX\_CASE 100
- Square number of the board array.
- #define MAX\_DIM 10
- Board array dimension.
- #define MAX\_MOVE 60
- Maximum number of moves during a game.
- #define BLACK 0
- Black pawn code.
- #define WHITE 1
- White pawn code.
- #define EMPTY 2
- Empty square code.
- #define BORDER 3
- Border square code.
- #define LEGAL 4
- Legal square code.
- #define TL\_CORNER 11
- Top left corner square code.
- #define TR\_CORNER 18
- Top right corner square code.
- #define BL\_CORNER 81
- bottom left corner square code
- #define BR\_CORNER 88
- bottom right corner square code
- #define TLD\_DIR -11
- Top left diagonal direction.
- #define TC\_DIR -10
- Top center direction.

- #define TRD\_DIR -9  
*Top right diagonal direction.*
- #define L\_DIR -1  
*Left direction.*
- #define R\_DIR +1  
*Right direction.*
- #define BLD\_DIR +9  
*Bottom left diagonal direction.*
- #define BC\_DIR +10  
*Bottom center diagonal direction.*
- #define BRD\_DIR +11  
*Bottom right diagonal direction.*
- #define PLAYER\_NB 2  
*Players number.*
- #define HUMAN 0  
*Human player type.*
- #define COMPUTER 1  
*Computer player type (AI)*
- #define H\_VS\_H 1  
*Human vs Human return code.*
- #define H\_VS\_C 2  
*Human vs Computer return code.*
- #define C\_VS\_C 3  
*Computer vs Computer return code.*
- #define EARLY\_GAME 10  
*Number of moves under which the game is considered as early.*
- #define MIDDLE\_GAME 42  
*Number of moves under which the game is considered as middle.*
- #define LATE\_GAME 60  
*Number of moves under which the game is considered as late.*
- #define MAX\_NAME 15  
*Maximum length of the player name string.*
- #define MAX\_LINE 15  
*Maximum length of the interpreter command string.*
- #define MAX\_FILE\_NAME 20  
*Maximum length of the saving file name string.*
- #define MAX\_FLIP 8  
*Size of the array for the existence of flip around a square of the board.*
- #define NB\_DIM 2  
*Game board dimensions number.*
- #define ASCII\_A 65  
*ASCII code of the 'A' letter.*
- #define ASCII\_H 72  
*ASCII code of the 'H' letter.*
- #define ASCII\_Z 90  
*ASCII code of the 'Z' letter.*
- #define ASCII\_A\_MIN 97  
*ASCII code of the 'a' letter.*
- #define ASCII\_Z\_MIN 122  
*ASCII code of the 'z' letter.*
- #define ASCII\_O 48



- ASCII code of the '0' number.*
  - #define `ASCII_8` 56
- ASCII code of the '8' number.*
  - #define `ASCII_9` 57
- ASCII code of the '9' number.*
  - #define `NL` 10
- New Line ASCII code.*
  - #define `SPACE` 32
- Space ASCII code.*
  - #define `LETTER` 1
- Return code for being a letter for a character.*
  - #define `NUMBER` 2
- Return code for being a number for a character.*
  - #define `BAD_COORDINATES` -1
- Return code for bad coordinates.*
  - #define `NOT_LEGAL` -2
- Return code for a not legal move.*
  - #define `UNKNOWN` '\0'
- Macro for the definition of players names before knowing it.*
  - #define `UNDEFINED` -1
- Macro for undefined initialisation.*
  - #define `DONT_TOUCH` -3
- Macro for not modifying some values. Used during a call to a function not to modify the value concerned.*
  - #define `END_OF_GAME` -1
- Return code for the end of the game.*
  - #define `COMP_NAME` "COMPUTER"
- The name of the computer during a human vs computer game.*
  - #define `ALT_COMP_NAME` "COMPUTER-IA"
- The alternative name of the computer during a human vs computer game.*

### 5.3.1 Detailed Description

A file in which one can find all the macros used in the Othello Project

### 5.3.2 Macro Definition Documentation

#### 5.3.2.1 #define GEN\_ALGO 1

Defines if the genetic algorithm is running or not.

#### Warning

Set to 0 for a use by real players, 1 to make the genetic algorithm run well

## 5.4 events.c File Reference

```
#include "events.h"
```

## Functions

- void `capture_SIGINT` ()  
*This function allows the program to redirect a ^C action (SIGINT signal) in order to display a short message informing them the program will quit and the game will not be saved.*
- void `event_cancel` (Game \*game)  
*This function allows the program to cancel the last move upon the resquest of the user.*
- void `event_uncancel` (Game \*game)  
*This function allows the program to uncanceled the last move upon the resquest of the user.*
- void `event_help` ()  
*This function allows the program to display the help upon the resquest of the user.*
- Game \* `event_load` (Game \*game)  
*This function allows the program to load a game from a text file upon the resquest of the user.*
- void `event_save` (Game \*game)  
*This function allows the program to save the current game upon the resquest of the user.*
- Game \* `event_new` (Game \*game)  
*This function allows the program to create a new game upon the resquest of the user.*
- Game \* `event_other` (Game \*game, char \*command)  
*This function allows the program to play the coordinates the user has entered if it was not one of the recognized commands.*
- void `event_quit` (Game \*game)  
*This function allows the program to quit properly, by freeing the memory.*

### 5.4.1 Detailed Description

File for the management of the different events the game can receive

### 5.4.2 Function Documentation

#### 5.4.2.1 void event\_cancel ( Game \* game )

This function allows the program to cancel the last move upon the resquest of the user.

##### Parameters

<code>game</code>	: pointer to the <code>Game</code> structure
-------------------	--

Tests if a game is in progress. If so, cancels 1 move if it is a Human Vs Human game, 2 for a Human Vs Computer game. Clears the screen and displays the game board.

#### 5.4.2.2 void event\_help ( )

This function allows the program to display the help upon the resquest of the user.

Clears the current screen and displays the help.

#### 5.4.2.3 Game\* event\_load ( Game \* game )

This function allows the program to load a game from a text file upon the resquest of the user.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

## Returns

the pointer to the modified game

Tests if a game is in progress. If so, asks for its saving and destroys the current game. Loads a new game, displays the board and the turn if the end of game has not been reached. Performs an end of game action otherwise and returns the pointer to the new game.

5.4.2.4 `Game* event_new ( Game * game )`

This function allows the program to create a new game upon the request of the user.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

## Returns

the pointer to the modified game

If a game is already in progress, asks for its saving and destroys it. Initializes a new game, asks for the type of the game, asks for the names and the colors, displays the board and the turn. Calls [play\\_ai\(\)](#) if the computer needs to play. returns the new pointer to the game.

5.4.2.5 `Game* event_other ( Game * game, char * command )`

This function allows the program to play the coordinates the user has entered if it was not one of the recognized commands.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
<i>command</i>	: the command executed in the main, which is supposed to be the coordinate of the move

## Returns

the pointer to the modified game

For the game in progress, tries to play the move related to the coordinates entered. If they don't have the proper format, returns an error code. Same for non legal coordinates. Once the move played, performs an end of game action if needed. Displays the turn and makes the computer play if needed. Returns the game if modified by [play\\_ai\(\)](#).

5.4.2.6 `void event_quit ( Game * game )`

This function allows the program to quit properly, by freeing the memory.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

This function tests if the current game was saved or not. If not so, it asks the user for it and demands a name for the backup file. That done, it frees the previous allocated memory and exits the program with the EXIT\_SUCCESS code.

#### 5.4.2.7 void event\_save ( **Game** \* *game* )

This function allows the program to save the current game upon the request of the user.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

Tests if a game is in progress. If so, asks for its saving.

5.4.2.8 void event\_uncancel ( [Game](#) \* *game* )

This function allows the program to uncanceled the last move upon the request of the user.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

Tests if a game is in progress. If so, uncanceled 1 move if it is a Human Vs Human game, 2 for a Human Vs Computer game. Clears the screen and displays the game board.

## 5.5 events.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include "ai.h"
#include "constants.h"
#include "graphics.h"
#include "io.h"
#include "othellier.h"
#include "structures.h"
```

## Functions

- void [capture\\_SIGINT](#) ()
 

*This function allows the program to redirect a ^C action (SIGINT signal) in order to display a short message informing them the program will quit and the game will not be saved.*
- void [event\\_cancel](#) ([Game](#) \*game)
 

*This function allows the program to cancel the last move upon the request of the user.*
- void [event\\_uncancel](#) ([Game](#) \*game)
 

*This function allows the program to uncanceled the last move upon the request of the user.*
- void [event\\_help](#) ()
 

*This function allows the program to display the help upon the request of the user.*
- [Game](#) \* [event\\_load](#) ([Game](#) \*game)
 

*This function allows the program to load a game from a text file upon the request of the user.*
- void [event\\_save](#) ([Game](#) \*game)
 

*This function allows the program to save the current game upon the request of the user.*
- [Game](#) \* [event\\_new](#) ([Game](#) \*game)
 

*This function allows the program to create a new game upon the request of the user.*
- [Game](#) \* [event\\_other](#) ([Game](#) \*game, char \*command)
 

*This function allows the program to play the coordinates the user has entered if it was not one of the recognized commands.*
- void [event\\_quit](#) ([Game](#) \*game)
 

*This function allows the program to quit properly, by freeing the memory.*

### 5.5.1 Detailed Description

The header file of the [events.c](#) file. Here you can find the prototypes of all functions of this file

### 5.5.2 Function Documentation

#### 5.5.2.1 void event\_cancel ( Game \* game )

This function allows the program to cancel the last move upon the request of the user.

##### Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

Tests if a game is in progress. If so, cancels 1 move if it is a Human Vs Human game, 2 for a Human Vs Computer game. Clears the screen and displays the game board.

#### 5.5.2.2 void event\_help ( )

This function allows the program to display the help upon the request of the user.

Clears the current screen and displays the help.

#### 5.5.2.3 Game\* event\_load ( Game \* game )

This function allows the program to load a game from a text file upon the request of the user.

##### Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

##### Returns

the pointer to the modified game

Tests if a game is in progress. If so, asks for its saving and destroys the current game. Loads a new game, displays the board and the turn if the end of game has not been reached. Performs an end of game action otherwise and returns the pointer to the new game.

#### 5.5.2.4 Game\* event\_new ( Game \* game )

This function allows the program to create a new game upon the request of the user.

##### Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

##### Returns

the pointer to the modified game

If a game is already in progress, asks for its saving and destroys it. Initializes a new game, asks for the type of the game, asks for the names and the colors, displays the board and the turn. Calls [play\\_ai\(\)](#) if the computer needs to play. returns the new pointer to the game.

#### 5.5.2.5 Game\* event\_other ( Game \* game, char \* command )

This function allows the program to play the coordinates the user has entered if it was not one of the recognized commands.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
<i>command</i>	: the command executed in the main, which is supposed to be the coordinate of the move

## Returns

the pointer to the modified game

For the game in progress, tries to play the move related to the coordinates entered. If they don't have the proper format, returns an error code. Same for non legal coordinates. Once the move played, performs and end of game action if needed. Displays the turn and makes the computer play if needed. Returns the game if modified by [play\\_ai\(\)](#).

## 5.5.2.6 void event\_quit ( Game \* game )

This function allows the program to quit properly, by freeing the memory.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

This function tests if the current game was saved or not. If not so, it asks the user for it and demands a name for the backup file. That done, it frees the previous allocated memory and exits the program with the EXIT\_SUCCESS code.

## 5.5.2.7 void event\_save ( Game \* game )

This function allows the program to save the current game upon the request of the user.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

Tests if a game is in progress. If so, asks for its saving.

## 5.5.2.8 void event\_uncancel ( Game \* game )

This function allows the program to uncanceled the last move upon the request of the user.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

Tests if a game is in progress. If so, uncanceled 1 move if it is a Human Vs Human game, 2 for a Human Vs Computer game. Clears the screen and displays the game board.

## 5.6 game.c File Reference

```
#include "game.h"
```

## Functions

- [Game \\* init\\_game \(\)](#)  
*This function initializes the [Game](#) structure.*
- void [load\\_coefs](#) (float coefs[])  
*This functions loads and/or initializes the coefficients for the evaluation function.*

- void `update_turn` (`Game` \*game)  
*This function allows to update the field showing the player who has the turn.*
- short `whose_turn_is_it` (`Game` \*game, int color)  
*This function determines which player has the turn.*
- short `has_legal_move` (`Game` \*game, int color)  
*This function determines if the player passed in argument is still able to play on the board.*
- void `destroy_game` (`Game` \*game)  
*This function destroy the `Game` structure.*
- short `destroy_end_moves_tab` (`Game` \*game)  
*This function frees the moves array from a cursor to another. It is used especially for freeing the end of the array, when a new move is played (after a cancellation).*
- short `end_of_game` (`Game` \*game)  
*This function determines whether the game is over or not.*
- short `is_computer_game` (`Game` \*game)  
*This function tells if the game in progress is a computer game or not.*
- `Game` \* `end_of_game_action` (`Game` \*game)  
*This function chooses the action to realize when the game reaches its end.*
- `Game` \* `play_ai` (`Game` \*game)  
*This function allows the AI to play a turn, if it has to.*

### 5.6.1 Detailed Description

File for the algorithms and functions related to the management of the game itself

### 5.6.2 Function Documentation

#### 5.6.2.1 short `destroy_end_moves_tab` ( `Game` \* *game* )

This function frees the moves array from a cursor to another. It is used especially for freeing the end of the array, when a new move is played (after a cancellation).

##### Parameters

<i>game</i>	: pointer to the <code>Game</code> structure
-------------	--

##### Returns

1 if at least 1 cell of the array has been freed, 0 otherwise

Tests if the cursors are equal or not. If not so, frees all cells of the array from the end to the cursor of the game. Reallocates a memory place for the new array (which is smaller).

#### 5.6.2.2 void `destroy_game` ( `Game` \* *game* )

This function destroy the `Game` structure.

Frees the played moves memory, the board (`Othellier` structure) and finally the `Game` structure

#### 5.6.2.3 short `end_of_game` ( `Game` \* *game* )

This function determines whether the game is over or not.



## Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

## Returns

YES if so, NO otherwise

Tests if the at least one player has no remaining pawn on the board. If so, the game is over. If not, tests if at least one player still have remaining possible moves. If not, the game is over, otherwise it is not.

5.6.2.4 **Game\*** end\_of\_game\_action ( **Game \*** *game* )

This function chooses the action to realize when the game reaches its end.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

## Returns

*game* : the new initialized pointer to the [Game](#) structure or the game pointer freed (NULL).

Tests whether the game has indeed reached its end. Then informs the players who has won and asks them for the saving of the game. Asks for a new game, and returns the pointer to the new game initialized, returns NULL otherwise.

5.6.2.5 **short** has\_legal\_move ( **Game \*** *game*, **int** *color* )

This function determines if the player passed in argument is still able to play on the board.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color of the player

## Returns

YES if the player still has possible moves, NO otherwise

After changing the turn to the color of the player, updates the state of the board, and for each square, tests if the move is legal. If so, changes again the turn and return YES, return NO otherwise.

5.6.2.6 **Game\*** init\_game ( )

This function initializes the [Game](#) structure.

## Returns

*game* : the initialized pointer to the [Game](#) structure

Allocates the game, initializes all the values needed for the game such as the players' names, their types, levels, the cursors and put the first turn to BLACK. Makes a call to the function [init\\_othellier\(\)](#) to initialize the [Othellier](#) structure

5.6.2.7 **short** is\_computer\_game ( **Game \*** *game* )

This function tells if the game in progress is a computer game or not.

**Parameters**

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

**Returns**

YES for a positive answer, NO otherwise

Tests if one of the two players is the computer

**5.6.2.8 void load\_coefs ( float *coefs*[ ] )**

This functions loads and/or initializes the coefficients for the evaluation function.

**Parameters**

<i>coefs</i>	: the coefficients array
--------------	--------------------------

Tests if the file containing the coefficients exists. If so, loads them from it. Otherwise, initializes them to 1.

**5.6.2.9 Game\* play\_ai ( Game \* *game* )**

This function allows the AI to play a turn, if it has to.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

**Returns**

the game structre, as it may very well be deallocated in this function (if the game reaches its end)

Tests if the turn is really to the computer (AI). Then, calls the [min\\_max\(\)](#) function to find the best move to play. Tests whether or not its the end of the game, and returns the game structure or perform an action of end of game if so.

**5.6.2.10 void update\_turn ( Game \* *game* )**

This function allows to update the field showing the player who has the turn.

**Parameters**

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

Makes a simple call to the function [whose\\_turn\\_is\\_it\(\)](#) to determine the player who has the turn. It changes the field turn in the [Game](#) structure

**5.6.2.11 short whose\_turn\_is\_it ( Game \* *game*, int *color* )**

This function determines which player has the turn.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color of the player who has just played

**Returns**

The color of the player who has the turn, END\_OF\_GAME otherwise

Tests if the supposed next player still has possible moves on the board. If so, he is the next player. If not, tests whither the ohter player still has possible moves. If so, he is the next player, if not, the game is over.

## 5.7 game.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "ai.h"
#include "graphics.h"
#include "othellier.h"
#include "structures.h"
#include "user.h"
```

### Functions

- [Game \\* init\\_game \(\)](#)  
*This function initializes the [Game](#) structure.*
- void [load\\_coefs](#) (float coefs[])  
*This functions loads and/or initializes the coefficients for the evaluation function.*
- void [update\\_turn](#) ([Game](#) \*game)  
*This function allows to update the field showing the player who has the turn.*
- short [whose\\_turn\\_is\\_it](#) ([Game](#) \*game, int color)  
*This function determines which player has the turn.*
- void [destroy\\_game](#) ([Game](#) \*game)  
*This function destroy the [Game](#) structure.*
- short [has\\_legal\\_move](#) ([Game](#) \*game, int color)  
*This function determines if the player passed in argument is still able to play on the board.*
- short [destroy\\_end\\_moves\\_tab](#) ([Game](#) \*game)  
*This function frees the moves array from a cursor to another. It is used especially for freeing the end of the array, when a new move is played (after a cancellation).*
- short [end\\_of\\_game](#) ([Game](#) \*game)  
*This function determines whether the game is over or not.*
- short [is\\_computer\\_game](#) ([Game](#) \*game)  
*This function tells if the game in progress is a computer game or not.*
- [Game \\* end\\_of\\_game\\_action](#) ([Game](#) \*game)  
*This function chooses the action to realize when the game reaches its end.*
- [Game \\* play\\_ai](#) ([Game](#) \*game)  
*This function allows the AI to play a turn, if it has to.*

### 5.7.1 Detailed Description

The header file of the [game.c](#) file. Here you cand find the prototypes of all functions of this file

### 5.7.2 Function Documentation

#### 5.7.2.1 short destroy\_end\_moves\_tab ( [Game](#) \* game )

This function frees the moves array from a cursor to another. It is used especially for freeing the end of the array, when a new move is played (after a cancellation).

**Parameters**

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

**Returns**

1 if at least 1 cell of the array has been freed, 0 otherwise

Tests if the cursors are equal or not. If not so, frees all cells of the array from the end to the cursor of the game. Reallocates a memory place for the new array (which is smaller).

**5.7.2.2 void destroy\_game ( Game \* game )**

This function destroy the [Game](#) structure.

Frees the played moves memory, the board ([Othellier](#) structure) and finally the [Game](#) structure

**5.7.2.3 short end\_of\_game ( Game \* game )**

This function determines whether the game is over or not.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

**Returns**

YES if so, NO otherwise

Tests if the at least one player has no remaining pawn on the board. If so, the game is over. If not, tests if at least one player still have remaining possible moves. If not, the game is over, otherwise it is not.

**5.7.2.4 Game\* end\_of\_game\_action ( Game \* game )**

This function chooses the action to realize when the game reaches its end.

**Parameters**

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

**Returns**

game : the new initialized pointer to the [Game](#) structure or the game pointer freed (NULL).

Tests whether the game has indeed reached its end. Then informs the players who has won and asks them for the saving of the game. Asks for a new game, and returns the pointer to the new game initialized, returns NULL otherwise.

**5.7.2.5 short has\_legal\_move ( Game \* game, int color )**

This function determines if the player passed in argument is still able to play on the board.

**Parameters**

--	--

<i>game</i>	: pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color of the player

**Returns**

YES if the player still has possible moves, NO otherwise

After changing the turn to the color of the player, updates the state of the board, and for each square, tests if the move is legal. If so, changes again the turn and return YES, return NO otherwise.

**5.7.2.6 Game\* init\_game ( )**

This function initializes the [Game](#) structure.

**Returns**

game : the initialized pointer to the [Game](#) structure

Allocates the game, initializes all the values needed for the game such as the players' names, their types, levels, the cursors and puts the first turn to BLACK. Makes a call to the function [init\\_othellier\(\)](#) to initialize the [Othellier](#) structure

**5.7.2.7 short is\_computer\_game ( Game \* game )**

This function tells if the game in progress is a computer game or not.

**Parameters**

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

**Returns**

YES for a positive answer, NO otherwise

Tests if one of the two players is the computer

**5.7.2.8 void load\_coefs ( float coefs[] )**

This function loads and/or initializes the coefficients for the evaluation function.

**Parameters**

<i>coefs</i>	: the coefficients array
--------------	--------------------------

Tests if the file containing the coefficients exists. If so, loads them from it. Otherwise, initializes them to 1.

**5.7.2.9 Game\* play\_ai ( Game \* game )**

This function allows the AI to play a turn, if it has to.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

**Returns**

the game structure, as it may very well be deallocated in this function (if the game reaches its end)

Tests if the turn is really to the computer (AI). Then, calls the [min\\_max\(\)](#) function to find the best move to play. Tests whether or not it's the end of the game, and returns the game structure or performs an action of end of game if so.

**5.7.2.10 void update\_turn ( Game \* *game* )**

This function allows to update the field showing the player who has the turn.

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

Makes a simple call to the function [whose\\_turn\\_is\\_it\(\)](#) to determine the player who has the turn. It changes the field turn in the [Game](#) structure

## 5.7.2.11 short whose\_turn\_is\_it ( Game \* game, int color )

This function determines which player has the turn.

## Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color of the player who has just played

## Returns

The color of the player who has the turn, END\_OF\_GAME otherwise

Tests if the supposed next player still has possible moves on the board. If so, he is the next player. If not, tests whether the other player still has possible moves. If so, he is the next player, if not, the game is over.

## 5.8 genetic.c File Reference

```
#include "genetic.h"
```

## Functions

- float [random\\_tenth](#) (short lowerBound, short upperBound)  
*This function generates a random number between lowerBound and upperBound.*
- [Population](#) \* [initial\\_population](#) (short genNb)  
*This function creates a random initial population.*
- void [fitness](#) ([Population](#) \*pop, short gameNb, FILE \*f1)  
*This function makes genomes play against each other.*
- [Population](#) \* [next\\_generation](#) ([Population](#) \*initial)  
*This function creates a new generation based on the previous one.*
- void [reproduction](#) ([Population](#) \*initial, [Population](#) \*next, short parentA, short parentB, short childA, short childB)  
*Allows two genomes to reproduce in order to create 2 new ones.*
- void [mutation](#) ([Population](#) \*next, short genomeToMutate)  
*Allows genomes to mutate.*
- short [search\\_best\\_genome](#) ([Population](#) \*pop)  
*Searches the best genome in the population in argument.*
- void [save\\_coefs](#) (float coefs[])  
*Saves the coefficients passed in argument in a file with a given filename.*
- void [destroy\\_population](#) ([Population](#) \*pop)  
*Frees the memory relative to the population in argument.*
- void [genetic\\_algorithm](#) ()  
*Gentic\_algorithm : main function.*

### 5.8.1 Detailed Description

Functions related to the genetic algorithm (used to determine the different coefficients of the evaluation function)

### 5.8.2 Function Documentation

#### 5.8.2.1 void destroy\_population ( Population \* pop )

Frees the memory relative to the population in argument.

Parameters

<i>pop</i>	: pointer to a population structure
------------	-------------------------------------

#### 5.8.2.2 void fitness ( Population \* pop, short gameNb, FILE \* f1 )

This function makes genomes play against each other.

Parameters

<i>pop</i>	: pointer to a population structure
<i>gameNb</i>	: the number of game each genome has to play
<i>f1</i>	: the file in which data are stored

Makes each genome play against gameNb others, caculates their fitness and stores data in the file.

#### 5.8.2.3 void genetic\_algorithm ( )

Gentic\_algorithm : main function.

Initializes a new population, makes each genome plays against k others, caculates their fitness, selects them using the biased wheel, makes them reproduce, mutates, and again over a defined number of generation.

#### 5.8.2.4 Population\* initial\_population ( short populationNb )

This function creates a random initial population.

Parameters

<i>populationNb</i>	: pointer to the <a href="#">Game</a> structure
---------------------	---

Returns

the initialized population

The allocates the memory space used for a population of size populationNb. Generates random coefficients and initilizes the fitness to 0

#### 5.8.2.5 void mutation ( Population \* next, short genomeToMutate )

Allows genomes to mutate.

Parameters

---



<i>next</i>	: pointer to the next population structure
<i>genomeTo-Mutate</i>	: the index of the genome to mutate in the next generation

Randomly chose a number between 0 and 1. If this one is below MUT\_FACTOR, changes randomly all coefficients of the genome.

#### 5.8.2.6 Population\* next\_generation ( Population \* initial )

This function creates a new generation based on the previous one.

##### Parameters

<i>initial</i>	: pointer to the initial population structure
----------------	---

##### Returns

the new generation created

Uses the system of the biased wheel in order to select most of the best genomes. Makes them reproduce to create a new generation, and makes each genome mutate.

#### 5.8.2.7 float random\_tenth ( short lowerBound, short upperBound )

This function generates a random number between lowerBound and upperBound.

##### Parameters

<i>lowerBound</i>	: the lower bound of the random number
<i>upperBound</i>	: the upper bound of the random number

##### Returns

a random float

Generates a random number between lowerBound and upperBound. Rounds it to the nearest tenth number.

#### 5.8.2.8 void reproduction ( Population \* initial, Population \* next, short parentA, short parentB, short childA, short childB )

Allows two genomes to reproduce in order to create 2 new ones.

##### Parameters

<i>initial</i>	: pointer to the initial population structure
<i>next</i>	: pointer to the next population structure
<i>parentA</i>	: number of the first parent
<i>parentB</i>	: number of the second parent
<i>childA</i>	: number of the first child
<i>childB</i>	: number of the second child

Uses random numbers to make 2 genomes reproduce and stores the result in the next population at indexes childA and childB.

#### 5.8.2.9 void save\_coefs ( float coefs[] )

Saves the coefficients passed in argument in a file with a given filename.

**Parameters**

<code>coefs[]</code>	: the coefficients to save
----------------------	----------------------------

Opens the file and saves each coefficient in it.

**5.8.2.10 short search\_best\_genome ( Population \* pop )**

Searches the best genome in the population in argument.

**Parameters**

<code>pop</code>	: pointer to a population structure
------------------	-------------------------------------

**Returns**

the index of the ebst genome found

**5.9 genetic.h File Reference**

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include "constants.h"
#include "structures.h"
#include "ai.h"
#include "move.h"
```

**Functions**

- float [random\\_tenth](#) (short lowerBound, short upperBound)  
*This function generates a random number between lowerBound and upperBound.*
- [Population \\* initial\\_population](#) (short populationNb)  
*This function creates a random initial population.*
- void [fitness](#) ([Population \\*pop](#), short gameNb, FILE \*f1)  
*This function makes genomes play against each other.*
- [Population \\* next\\_generation](#) ([Population \\*initial](#))  
*This function creates a new generation based on the previous one.*
- void [reproduction](#) ([Population \\*initial](#), [Population \\*next](#), short parentA, short parentB, short childA, short childB)  
*Allows two genomes to reproduce in order to create 2 new ones.*
- void [mutation](#) ([Population \\*next](#), short genomeToMutate)  
*Allows genomes to mutate.*
- short [search\\_best\\_genome](#) ([Population \\*pop](#))  
*Searches the best genome in the population in argument.*
- void [save\\_coefs](#) (float coefs[])  
*Saves the coefficients passed in argument in a file with a given filename.*
- void [destroy\\_population](#) ([Population \\*pop](#))  
*Frees the memory relative to the population in argument.*
- void [genetic\\_algorithm](#) ()  
*Gentic\_algorithm : main function.*

### 5.9.1 Detailed Description

The header file of the .c file [genetic.c](#). Here you can find the prototypes of all functions of this file

### 5.9.2 Function Documentation

#### 5.9.2.1 void destroy\_population ( Population \* pop )

Frees the memory relative to the population in argument.

##### Parameters

<i>pop</i>	: pointer to a population structure
------------	-------------------------------------

#### 5.9.2.2 void fitness ( Population \* pop, short gameNb, FILE \* f1 )

This function makes genomes play against each other.

##### Parameters

<i>pop</i>	: pointer to a population structure
<i>gameNb</i>	: the number of game each genome has to play
<i>f1</i>	: the file in which data are stored

Makes each genome play against gameNb others, caculates their fitness and stores data in the file.

#### 5.9.2.3 void genetic\_algorithm ( )

Gentic\_algorithm : main function.

Initializes a new population, makes each genome plays against k others, caculates their fitness, selects them using the biased wheel, makes them reproduce, mutates, and again over a defined number of generation.

#### 5.9.2.4 Population\* initial\_population ( short populationNb )

This function creates a random initial population.

##### Parameters

<i>populationNb</i>	: pointer to the <a href="#">Game</a> structure
---------------------	---

##### Returns

the initialized population

The allocates the memory space used for a population of size populationNb. Generates random coefficients and initilizes the fitness to 0

#### 5.9.2.5 void mutation ( Population \* next, short genomeToMutate )

Allows genomes to mutate.

##### Parameters

<i>next</i>	: pointer to the next population structure
<i>genomeTo-Mutate</i>	: the index of the genome to mutate in the next generation

Randomly chose a number between 0 and 1. If this one is below MUT\_FACTOR, changes randomly all coefficients of the genome.

#### 5.9.2.6 **Population\* next\_generation ( Population \* initial )**

This function creates a new generation based on the previous one.

##### Parameters

<i>initial</i>	: pointer to the initial population structure
----------------	---

##### Returns

the new generation created

Uses the system of the biased wheel in order to select most of the best genomes. Makes them reproduce to create a new generation, and makes each genome mutate.

#### 5.9.2.7 **float random\_tenth ( short lowerBound, short upperBound )**

This function generates a random number between lowerBound and upperBound.

##### Parameters

<i>lowerBound</i>	: the lower bound of the random number
<i>upperBound</i>	: the upper bound of the random number

##### Returns

a random float

Generates a random number between lowerBound and upperBound. Rounds it to the nearest tenth number.

#### 5.9.2.8 **void reproduction ( Population \* initial, Population \* next, short parentA, short parentB, short childA, short childB )**

Allows two genomes to reproduce in order to create 2 new ones.

##### Parameters

<i>initial</i>	: pointer to the initial population structure
<i>next</i>	: pointer to the next population structure
<i>parentA</i>	: number of the first parent
<i>parentB</i>	: number of the second parent
<i>childA</i>	: number of the first child
<i>childB</i>	: number of the second child

Uses random numbers to make 2 genomes reproduce and stores the result in the next population at indexes childA and childB.

#### 5.9.2.9 **void save\_coefs ( float coefs[] )**

Saves the coefficients passed in argument in a file with a given filename.

## Parameters

<code>coefs[]</code>	: the coefficients to save
----------------------	----------------------------

Opens the file and saves each coefficient in it.

5.9.2.10 short search\_best\_genome ( Population \* *pop* )

Searches the best genome in the population in argument.

## Parameters

<code>pop</code>	: pointer to a population structure
------------------	-------------------------------------

## Returns

the index of the ebst genome found

## 5.10 graphics.c File Reference

```
#include "graphics.h"
```

## Functions

- void [reinit\\_graphics](#) ()  
*This function reinitializes the graphics (black background and white text by default)*

## 5.10.1 Detailed Description

Functions related to the developement of the graphics of the game

## Warning

Not fully implemented for compatibility purpose

## 5.11 graphics.h File Reference

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

## Macros

- #define [clear\\_screen](#)() printf("\033[H\033[2J")  
*Preprocessor directive for the clearing of the console screen.*
- #define [color](#)(param) printf("\033[%sm", param)  
*Preprocessor directive for changing the text color and/or the background color.*
- #define [BLACK\\_TEXT](#) "30"  
*Value for a text in black.*
- #define [RED\\_TEXT](#) "31"  
*Value for a text in red.*

- `#define GREEN_TEXT "32"`  
*Value for a text in green.*
- `#define YELLOW_TEXT "33"`  
*Value for a text in yellow.*
- `#define BLUE_TEXT "34"`  
*Value for a text in blue.*
- `#define MAGENTA_TEXT "35"`  
*Value for a text in magenta.*
- `#define CYAN_TEXT "36"`  
*Value for a text in cyan.*
- `#define WHITE_TEXT "37"`  
*Value for a text in white.*
- `#define BLACK_BOARD "40"`  
*Value for a background in black.*
- `#define RED_BOARD "41"`  
*Value for a background in red.*
- `#define GREEN_BOARD "42"`  
*Value for a background in green.*
- `#define YELLOW_BOARD "43"`  
*Value for a background in yellow.*
- `#define BLUE_BOARD "44"`  
*Value for a background in blue.*
- `#define MAGENTA_BOARD "45"`  
*Value for a background in magenta.*
- `#define CYAN_BOARD "46"`  
*Value for a background in cyan.*
- `#define WHITE_BOARD "47"`  
*Value for a background in white.*
- `#define BOLD_TEXT "1"`  
*Value for a text in bold.*
- `#define REINIT "0"`  
*Value to reinitialize the background and text color.*

## Functions

- `void reinit_graphics ()`  
*This function reinitializes the graphics (black background and white text by default)*

### 5.11.1 Detailed Description

The header file of the `graphics.c` file. Here you can find the prototypes of all functions of this file, and the macros related to the graphics of the game

## 5.12 io.c File Reference

```
#include "io.h"
```

## Functions

- void `display_othellier` (`Game *game`)  
*This function displays the game board based on the type of the square.*
- char \* `int2str` (int pos)  
*This function converts an integer (representing a position in an array (max. size : MAX\_CASE)) to a string containing coordinates (letter of the column, number of the row)*
- int `char2int` (const char \*pos)  
*This function converts a string containing coordinates (letter of the column, number of the row) to an integer (representing a position in an array (max. size : MAX\_CASE))*
- int `are_good_coordinates` (const char \*coord)  
*This function determines if the coordinates passed in argument have the proper format to be converted to an integer.*
- int `game_saving` (`Game *game`, const char \*filename)  
*This function allows the user to save the game in progress.*
- `Game *` `load_game` (char \*filename)  
*This function imports/loads a game from a text file and store the data int the `Game` structure.*
- int `is_a_letter` (char c)  
*This function determines if the character passed in argument is a letter, a number, or something else.*
- int `contains_only_letters` (char \*str)  
*This function determines if the string passed in argument contains only letter and number or not.*
- int `str2upper` (char \*str)  
*This function modifies the string in argument to put each character in uppercase.*
- void `empty_buffer` ()  
*This function empty the buffer so that the remaining character in the stream (stdin) can not be used for a future retrieving of data.*
- int `exists_file` (char \*filename, int right)  
*This function determines if the file passed in argument can be opened with the rights passed in argument.*
- int `ask_YN` (char \*string)  
*This function allows to ask a YES/NO question to the user with the string passed in argument.*

### 5.12.1 Detailed Description

Functions related to the processing of Input / Output data

### 5.12.2 Function Documentation

#### 5.12.2.1 int are\_good\_coordinates ( const char \* coord )

This function determines if the coordinates passed in argument have the proper format to be converted to an integer.

##### Parameters

<code>coord</code>	: a string containing the coordinates
--------------------	---------------------------------------

##### Returns

Yes if the coordinates are in the good format, NO otherwise

If the column is 'under' A or 'over' H, the coordinates aren't good. The same way, if the line is under 0 or over 8, the coordinates are not in the board.

#### 5.12.2.2 int ask\_YN ( char \* string )

This function allows to ask a YES/NO question to the user with the string passed in argument.

**Parameters**

<i>string</i>	: the string to display as a Y/N question
---------------	---

**Returns**

YES if the user answer is YES, No otherwise

While the user answer is not Y or N, the function displays the string followed by the item (Y/N) to inform the user the nature of the question. Each loop turn, it empty the buffer and transforms the answer in uppercase.

**5.12.2.3 int char2int ( const char \* pos )**

This function converts a string containing coordinates (letter of the column, number of the row) to an integer (representing a position in an array (max. size : MAX\_CASE))

**Parameters**

<i>pos</i>	: a string containing the coordinates
------------	---------------------------------------

**Returns**

an integer containing the coordinates converted

The function allocates converts the column letter into the tens digit of an integer, and the row number into the unit digit of the same integer.

**5.12.2.4 int contains\_only\_letters ( char \* str )**

This function determines if the string passed in argument contains only letter and number or not.

**Parameters**

<i>str</i>	: the string to analyze
------------	-------------------------

**Returns**

YES if it contains only letter or number, NO otherwise

For each character of the string, the function tests if it is a letter or a number. If so, it returns YES at the end. If a single character is not a letter or number, it returns NO.

**5.12.2.5 void display\_othellier ( Game \* game )**

This function displays the game board based on the type of the square.

**Parameters**

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

The function is made so that the display looks like a real board. It displays a square for a black pawn, a triangle for a white pawn, a circle for a legal move, and pipes and underscore to delimit the borders of the board.

**5.12.2.6 void empty\_buffer ( )**

This function empty the buffer so that the remaining character in the stream (stdin) can not be used for a future retrieving of data.

While the character is not a new line or EOF, the function calls fgetc and loses the data taken.



#### 5.12.2.7 int exists\_file ( char \* *filename*, int *right* )

This function determines if the file passed in argument can be opened with the rights passed in argument.

**Parameters**

<i>filename</i>	: the name of the file to open
<i>right</i>	: the right to test : F_OK (to test the existence of the file), R_OK (for the reading right) or W_OK (for the writing right)

**Returns**

YES if the file exists and has the rights required, NO otherwise

Calls the function access from <unistd.h> and determines the rights thanks to it.

**5.12.2.8 int game\_saving ( Game \* game, const char \* filename )**

This function allows the user to save the game in progress.

**Parameters**

<i>game</i>	: a pointer to theGame structure
<i>filename</i>	: a string containing the name of the file in which the user wants to save the game

**Returns**

1 if the game has been saved, exits if the file can not be open

Opens the file names as 'filename'. Writes the moves done during the game under the format [+][Column letter][Row number] for a black move or [-] for a white move. Writes the name of the black player, followed by his score, then the name of the white player followed by his score too. Updates the status of the 'saved game' box to YES.

**5.12.2.9 char\* int2str ( int pos )**

This function converts an integer (representing a position in an array (max. size : MAX\_CASE)) to a string containing coordinates (letter of the column, number of the row)

**Parameters**

<i>pos</i>	: a position integer
------------	----------------------

**Returns**

a string containing the coordinates converted

The function allocates a string with 3 character (column, line and the terminating character), converts into a line using the tens digit of the pos integer, converts into column using the unit digit of the pos integer, and returns the string containing the coordinates.

**5.12.2.10 int is\_a\_letter ( char c )**

This function determines if the character passed in argument is a letter, a number, or something else.

**Parameters**

<i>c</i>	: the character to determine
----------	------------------------------

**Returns**

LETTER if it is a letter, NUMBER if it is a number, NO otherwise

Tests the ASCII code of the character passed. If it is between 'A' and 'Z' or 'a' and 'z', it is a letter. If it is between '0' and '9', it is a number. Otherwise, it is neither.

#### 5.12.2.11 `Game*` `load_game` ( `char *` *filename* )

This function imports/loads a game from a text file and store the data int the `Game` structure.

**Parameters**

<i>filename</i>	: a string containing the name of the file used to load the game
-----------------	--

**Returns**

the pointer to the [Game](#) structure just loaded

Opens the file names as 'filename'. Initializes the [Game](#) structure and loads the succession of moves in the moves array. For each move loaded, the function plays the move to have the right game board at the end. The pieces numbers are recalculated each time. Loads and stores players' names in their allocated space, closes the file and returns the pointer to the game.

**5.12.2.12 int str2upper ( char \* str )**

This function modifies the string in argument to put each character in uppercase.

**Parameters**

<i>str</i>	: the string to modify
------------	------------------------

**Returns**

1 if the string has been transformed in uppercase, 0 if the string length is null

For each character of the string, the function calls toupper, which puts a character into its uppercase equivalent. If there is no character to analyze, the function returns 0.

**5.13 io.h File Reference**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include "constants.h"
#include "game.h"
#include "move.h"
#include "structures.h"
```

**Functions**

- void [display\\_othellier](#) ([Game](#) \*game)  
*This function displays the game board based on the type of the square.*
- char \* [int2str](#) (int pos)  
*This function converts an integer (representing a position in an array (max. size : MAX\_CASE)) to a string containing coordinates (letter of the column, number of the row)*
- int [char2int](#) (const char \*pos)  
*This function converts a string containing coordinates (letter of the column, number of the row) to an integer (representing a position in an array (max. size : MAX\_CASE))*
- int [are\\_good\\_coordinates](#) (const char \*coord)  
*This function determines if the coordinates passed in argument have the proper format to be converted to an integer.*
- int [game\\_saving](#) ([Game](#) \*game, const char \*filename)  
*This function allows the user to save the game in progress.*

- `Game * load_game (char *filename)`  
*This function imports/loads a game from a text file and store the data int the `Game` structure.*
- `int is_a_letter (char c)`  
*This function determines if the character passed in argument is a letter, a number, or something else.*
- `int contains_only_letters (char *str)`  
*This function determines if the string passed in argument contains only letter and number or not.*
- `int str2upper (char *str)`  
*This function modifies the string in argument to put each character in uppercase.*
- `void empty_buffer ()`  
*This function empty the buffer so that the remaining character in the stream (stdin) can not be used for a future retrieving of data.*
- `int exists_file (char *filename, int right)`  
*This function determines if the file passed in argument can be opened with the rights passed in argument.*
- `int ask_YN (char *string)`  
*This function allows to ask a YES/NO question to the user with the string passed in argument.*

### 5.13.1 Detailed Description

The header file of the `io.c` file. Here you cand find the protoypes of all functions of this file

### 5.13.2 Function Documentation

#### 5.13.2.1 `int are_good_coordinates ( const char * coord )`

This function determines if the coordinates passed in argument have the proper format to be converted to an integer.

##### Parameters

<code>coord</code>	: a string containing the coordinates
--------------------	---------------------------------------

##### Returns

Yes if the coordinates are in the good format, NO otherwise

If the column is 'under' A or 'over' H, the coordinates aren't good. The same way, if the line is under 0 or over 8, the coordinates are not in the board.

#### 5.13.2.2 `int ask_YN ( char * string )`

This function allows to ask a YES/NO question to the user with the string passed in argument.

##### Parameters

<code>string</code>	: the string to display as a Y/N question
---------------------	---

##### Returns

YES if the user answer is YES, No otherwise

While the user answer is not Y or N, the function displays the string followed by the item (Y/N) to inform the user the nature of the question. Each loop turn, it empty the buffer and transforms the answer in uppercase.

#### 5.13.2.3 `int char2int ( const char * pos )`

This function converts a string containing coordinates (letter of the column, number of the row) to an integer (representing a position in an array (max. size : MAX\_CASE))

**Parameters**

<i>pos</i>	: a string containing the coordinates
------------	---------------------------------------

**Returns**

an integer containing the coordinates converted

The function allocates converts the column letter into the tens digit of an integer, and the row number into the unit digit of the same integer.

**5.13.2.4 int contains\_only\_letters ( char \* *str* )**

This function determines if the string passed in argument contains only letter and number or not.

**Parameters**

<i>str</i>	: the string to analyze
------------	-------------------------

**Returns**

YES if it contains only letter or number, NO otherwise

For each character of the string, the function tests if it is a letter or a number. If so, it returns YES at the end. If a single character is not a letter or number, it returns NO.

**5.13.2.5 void display\_othellier ( Game \* *game* )**

This function displays the game board based on the type of the square.

**Parameters**

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

The function is made so that the display looks like a real board. It displays a square for a black pawn, a triangle for a white pawn, a circle for a legal move, and pipes and underscore to delimit the borders of the board.

**5.13.2.6 void empty\_buffer ( )**

This function empty the buffer so that the remaining character in the stream (stdin) can not be used for a future retrieving of data.

While the character is not a new line or EOF, the function calls fgetc and loses the data taken.

**5.13.2.7 int exists\_file ( char \* *filename*, int *right* )**

This function determines if the file passed in argument can be opened with the rights passed in argument.

**Parameters**

<i>filename</i>	: the name of the file to open
<i>right</i>	: the right to test : F_OK (to test the existence of the file), R_OK (for the reading right) or W_OK (for the writing right)

**Returns**

YES if the file exists and has the rights required, NO otherwise

Calls the function access from <unistd.h> and determines the rights thanks to it.

5.13.2.8 `int game_saving ( Game * game, const char * filename )`

This function allows the user to save the game in progress.

**Parameters**

<i>game</i>	: a pointer to theGame structure
<i>filename</i>	: a string containing the name of the file in which the user wants to save the game

**Returns**

1 if the game has been saved, exits if the file can not be open

Opens the file names as 'filename'. Writes the moves done during the game under the format [ + ][Column letter][Row number] for a black move or [-] for a white move. Writes the name of the black player, followed by his score, then the name of the white player followed by his score too. Updates the status of the 'saved game' box to YES.

**5.13.2.9 char\* int2str ( int pos )**

This function converts an integer (representing a position in an array (max. size : MAX\_CASE)) to a string containing coordinates (letter of the column, number of the row)

**Parameters**

<i>pos</i>	: a position integer
------------	----------------------

**Returns**

a string containing the coordinates converted

The function allocates a string with 3 character (column, line and the terminating character), converts into a line using the tens digit of the pos integer, converts into column using the unit digit of the pos integer, and returns the string containing the cordinates.

**5.13.2.10 int is\_a\_letter ( char c )**

This function determines if the character passed in argument is a letter, a number, or something else.

**Parameters**

<i>c</i>	: the character to determine
----------	------------------------------

**Returns**

LETTER if it is a letter, NUMBER if it is a number, NO otherwise

Tests the ASCII code of the character passed. If it is between 'A' and 'Z' or 'a' and 'z', it is a letter. If it is between 'à' or '9', it is a number. Otherwise, it is neither.

**5.13.2.11 Game\* load\_game ( char \* filename )**

This function imports/loads a game from a text file and store the data int the [Game](#) structure.

**Parameters**

<i>filename</i>	: a string containing the name of the file used to load the game
-----------------	--

**Returns**

the pointer to the [Game](#) structure just loaded

Opens the file names as 'filename'. Initializes the [Game](#) structure ans loads the succession of moves in the moves array. For each move loaded, the function play the move to have the righth game board at the end. The pieces numbers are recalculated each time. Loads and store players' names in their allocated space, closes the file and returns the pointer to the game.



## 5.13.2.12 int str2upper ( char \* str )

This function modifies the string in argument to put each character in uppercase.

## Parameters

<i>str</i>	: the string to modify
------------	------------------------

## Returns

1 if the string has been transformed in uppercase, 0 if the string length is null

For each character of the string, the function calls toupper, which puts a character into its uppercase equivalent. If there is no character to analyze, the function return 0.

## 5.14 main.c File Reference

```
#include "main.h"
```

## Functions

- int [main](#) (void)

*The main function of the project.*

## 5.14.1 Detailed Description

The main file of the project. Here is only the main fonction

## 5.14.2 Function Documentation

## 5.14.2.1 int main ( void )

The main function of the project.

The main function has been designed to be a kind of interpreter. Almost at any time the user can make an action, such as loading a game, saving the current one, cancelling a move or redo one. String for the input in the interpreter (main). Maximum length MAX\_LINE

String for only the first word of the input in the interpreter (main). Maximum length MAX\_LINE

Temporary string (buffer) for the remaining characters of the input. Won't be used, only to extract the first word considered as the command

Integer to retrieve the number of arguments of line, but used to test if there was no input failure (EOF returned by sscanf)

## 5.15 main.h File Reference

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

```
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <string.h>
#include "constants.h"
#include "events.h"
#include "genetic.h"
#include "graphics.h"
#include "structures.h"
#include "user.h"
```

## Functions

- int [main](#) (void)

*The main function of the project.*

### 5.15.1 Detailed Description

The header file of the [main.c](#) file. Here you can find all the header requisites of the project, and the global variables

### 5.15.2 Function Documentation

#### 5.15.2.1 int main ( void )

The main function of the project.

The main function has been designed to be a kind of interpreter. Almost at any time the user can make an action, such as loading a game, saving the current one, cancelling a move or redo one. String for the input in the interpreter (main). Maximum length MAX\_LINE

String for only the first word of the input in the interpreter (main). Maximum length MAX\_LINE

Temporary string (buffer) for the remaining characters of the input. Won't be used, only to extract the first word considered as the command

Integer to retrieve the number of arguments of line, but used to test if there was no input failure (EOF returned by scanf)

## 5.16 move.c File Reference

```
#include "move.h"
```

## Functions

- [Move](#) \* [create\\_move](#) ([Othellier](#) \*oth, short position, short [color](#))

*This function creates a new [Move](#) using the parameters.*

- int [play\\_move](#) ([Game](#) \*game, [Move](#) \*mov)

*This function makes a move action on the game board.*

- int [move\\_processing](#) ([Game](#) \*game, const char \*coordinates, short intCoord)

*This function tests if the coordinates of a move are legal, and plays the move corresponding if so.*

- int [cancel\\_last\\_move](#) ([Game](#) \*game)

*This function cancel the last move played on the game board. If a move has already been canceled and there is still moves that can be canceled, the function does it as well.*

- int `uncancel_move` (`Game *game`)

*This function uncancel the last move canceled on the game board. If a move has already been uncanceled and there is still moves that can be uncanceled, the function does it as well.*

### 5.16.1 Detailed Description

Functions related to the processing of a move during the game

### 5.16.2 Function Documentation

#### 5.16.2.1 int `cancel_last_move` ( `Game * game` )

This function cancel the last move played on the game board. If a move has already been canceled and there is still moves that can be canceled, the function does it as well.

##### Parameters

<code>game</code>	: a pointer to the <code>Game</code> structure
-------------------	--

##### Returns

1 if the last move has been canceled, 0 if there was no move to cancel

Tests whether there is still moves that can be canceled. If so, decrements the game cursor and recalculates the board from the beginning with all moves in the moves array, except the canceled ones

#### 5.16.2.2 `Move*` `create_move` ( `Othellier * oth`, short `position`, short `color` )

This function creates a new `Move` using the parameters.

##### Parameters

<code>oth</code>	a pointer to the game board ( <code>Othellier</code> structure)
<code>position</code>	: the position of the move to initialize
<code>color</code>	: the color of the move to initialize

##### Returns

a pointer to the created move

The function allocates a new `Move` structure and fills it with the parameters in argument. Fills the flip array calling `update_flip()`, and returns the pointer.

#### 5.16.2.3 int `move_processing` ( `Game * game`, const char \* `coordinates`, short `intCoord` )

This function tests if the coordinates of a move are legal, and plays the move corresponding if so.

##### Parameters

<code>game</code>	: pointer to the <code>Game</code> structure
-------------------	--

<i>coordinates</i>	: string containing 2 characters : the first one must be the letter of the column and the second must be the number of the row where the player wants to play
<i>intCoord</i>	: the coordinates passed as an integer. If intCoord is UNDEFINED, the function uses the string coordinates. Otherwise, it uses the integer

#### Returns

1 if the move as been played, BAD\_COORDINATES if the coordinates don't have the right format, and NOT\_LEGAL if the move is not legal

Tests if the coordinates are passed as a string or as an integer and uses the right format. Tests if the coordinates have the proper format, allocates and initializes a move, tests if this move with the position is legal, frees the end of the moves array to prohibit the cancellation of old canceled moves, plays the move, reallocates the moves array and update the game status (game board and game turn).

#### 5.16.2.4 int play\_move ( Game \* game, Move \* mov )

This function makes a move action on the game board.

##### Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>mov</i>	: à pointer to a <a href="#">Move</a> action

#### Returns

The number of flipped pawns on the game board

The function begins with adding a pawn on the desired position on the game board. Then, it increases the pawn number of the current player. Plus, for each direction, if there is a flip, it returns the pawns that have the opposite color and increases the pawn number too.

#### 5.16.2.5 int uncanceled\_move ( Game \* game )

This function uncanceled the last move canceled on the game board. If a move has already been uncanceled and there is still moves that can be uncanceled, the function does it as well.

##### Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

#### Returns

1 if the last move has been uncanceled, 0 if there was no move to uncanceled

Tests whether there is still moves that can be uncanceled. If so, increments the game cursor and plays the last move canceled

## 5.17 move.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "constants.h"
#include "io.h"
#include "othellier.h"
#include "structures.h"
```

## Functions

- **Move \* create\_move** (**Othellier** \*oth, short position, short color)  
*This function creates a new **Move** using the parameters.*
- int **play\_move** (**Game** \*game, **Move** \*mov)  
*This function makes a move action on the game board.*
- int **move\_processing** (**Game** \*game, const char \*coordinates, short intCoord)  
*This function tests if the coordinates of a move are legal, and plays the move corresponding if so.*
- int **cancel\_last\_move** (**Game** \*game)  
*This function cancel the last move played on the game board. If a move has already been canceled and there is still moves that can be canceled, the function does it as well.*
- int **uncancel\_move** (**Game** \*game)  
*This function uncancel the last move canceled on the game board. If a move has already been uncanceled and there is still moves that can be uncanceled, the function does it as well.*

### 5.17.1 Detailed Description

The header file of the [move.c](#) file. Here you can find the prototypes of all functions of this file

### 5.17.2 Function Documentation

#### 5.17.2.1 int cancel\_last\_move ( **Game** \* game )

This function cancel the last move played on the game board. If a move has already been canceled and there is still moves that can be canceled, the function does it as well.

##### Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

##### Returns

1 if the last move has been canceled, 0 if there was no move to cancel

Tests whether there is still moves that can be canceled. If so, decrements the game cursor and recalculates the board from the beginning with all moves in the moves array, except the canceled ones

#### 5.17.2.2 **Move**\* create\_move ( **Othellier** \* oth, short position, short color )

This function creates a new [Move](#) using the parameters.

##### Parameters

<i>oth</i>	a pointer to the game board ( <a href="#">Othellier</a> structure)
<i>position</i>	: the position of the move to initialize
<i>color</i>	: the color of the move to initialize

##### Returns

a pointer to the created move

The function allocates a new [Move](#) structure and fills it with the parameters in argument. Fills the flip array calling [update\\_flip\(\)](#), and returns the pointer.

#### 5.17.2.3 int move\_processing ( **Game** \* game, const char \* coordinates, short intCoord )

This function tests if the coordinates of a move are legal, and plays the move corresponding if so.

**Parameters**

<i>game</i>	: pointer to the <a href="#">Game</a> structure
<i>coordinates</i>	: string containing 2 characters : the first one must be the letter of the column and the second must be the number of the row where the player wants to play
<i>intCoord</i>	: the coordinates passed as an integer. If intCoord is UNDEFINED, the function uses the string coordinates. Otherwise, it uses the integer

**Returns**

1 if the move as been played, BAD\_COORDINATES if the coordinates don't have the right format, and NOT\_LEGAL if the move is not legal

Tests if the coordinates are passed as a string or as an integer and uses the right format. Tests if the coordinates have the proper format, allocates and initializes a move, tests if this move with the position is legal, frees the end of the moves array to prohibit the cancellation of old canceled moves, plays the move, reallocates the moves array and update the game status (game board and game turn).

**5.17.2.4 int play\_move ( Game \* game, Move \* mov )**

This function makes a move action on the game board.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>mov</i>	: à pointer to a <a href="#">Move</a> action

**Returns**

The number of flipped pawns on the game board

The function begins with adding a pawn on the desired position on the game board. Then, it increases the pawn number of the current player. Plus, for each direction, if there is a flip, it returns the pawns that have the opposite color and increases the pawn number too.

**5.17.2.5 int uncanceled\_move ( Game \* game )**

This function uncanceled the last move canceled on the game board. If a move has already been uncanceled and there is still moves that can be uncanceled, the function does it as well.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

**Returns**

1 if the last move has been uncanceled, 0 if there was no move to uncanceled

Tests whether there is still moves that can be uncanceled. If so, increments the game cursor and plays the last move canceled

**5.18 othellier.c File Reference**

```
#include "othellier.h"
```

## Functions

- [Othellier](#) \* [init\\_othellier](#) ()  
*Initialisation de l'Othellier.*
- int [is\\_legal](#) ([Othellier](#) \*oth, [Move](#) \*mov)  
*This function determines whether the move passed in argument is a legal move on the game board passed in argument too.*
- int [update\\_flip](#) ([Othellier](#) \*oth, [Move](#) \*mov)  
*This function updates the status of each cell of the flip array for a move. If there is an alignment, the cell is set to 1, 0 otherwise.*
- short [has\\_alignment](#) ([Othellier](#) \*oth, short position, short [color](#), short direction)  
*This function tests the validity and the existence of an alignment in the given direction (see direction Macros)*
- int [update\\_othellier](#) ([Game](#) \*game)  
*This function updates the status of the game board, and shows for each empty square if it is a legal move for the user who has the turn, or not.*
- void [game\\_board\\_copy](#) ([Game](#) \*game)  
*This function copies the current game board state into the game board array (for saving)*
- int [destroy\\_end\\_boards\\_tab](#) ([Game](#) \*game)  
*This function frees the boards array from a cursor to another. It is used especially for freeing the end of the array, when a new move is played (after a cancellation).*

### 5.18.1 Detailed Description

Functions related to the game board, such as alignements and flips

### 5.18.2 Function Documentation

#### 5.18.2.1 int [destroy\\_end\\_boards\\_tab](#) ( [Game](#) \* *game* )

This function frees the boards array from a cursor to another. It is used especially for freeing the end of the array, when a new move is played (after a cancellation).

##### Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

##### Returns

1 if at least 1 cell of the array has been freed, 0 otherwise

Tests if the cursors are equal or not. If not so, frees all cells of the array from the end to the cursor of the game. Reallocates a memory place for the new array (which is smaller).

#### 5.18.2.2 void [game\\_board\\_copy](#) ( [Game](#) \* *game* )

This function copies the current game board state into the game board array (for saving)

##### Parameters

<i>game</i>	: a pointer to the game structure
-------------	-----------------------------------

Allocates a bigger space to add the new board. Copies material, turn and each squarre of the board. Increments the game board cursor.

#### 5.18.2.3 short [has\\_alignment](#) ( [Othellier](#) \* *oth*, short *pos*, short *color*, short *direction* )

This function tests the validity and the existence of an alignment in the given direction (see direction Macros)

**Parameters**

<i>oth</i>	: a pointer to the game board <a href="#">Othellier</a> structure
<i>pos</i>	: the position where the function needs to check for an alignment
<i>color</i>	: the color that the function needs to find at the end of the alignment to validate it
<i>direction</i>	: the direction of the alignment

**Returns**

1 if there is a valid alignment, 0 otherwise

While the square in the given direction is the opponent's color, tests the next position in the same direction. If it's the needed color, returns 1, continues the loop otherwise, until the position has another value than the opponent's color.

**5.18.2.4 Othellier\* init\_othellier ( )**

Initialisation de l'[Othellier](#).

This function initializes the main structure of the game board, so that one can begin a new game.

**5.18.2.5 int is\_legal ( Othellier \* oth, Move \* mov )**

This function determines whether the move passed in argument is a legal move on the game board passed in argument too.

**Parameters**

<i>oth</i>	a pointer to the game board
<i>mov</i>	: a pointer to a <a href="#">Move</a> action

**Returns**

YES if the move is legal, NO otherwise

The function tests if the move is legal by testing the value of the square on the game board. If the square is not empty, the move cannot be legal. Then, it tests for each cell of the flip array if there is any legal alignment

**5.18.2.6 int update\_flip ( Othellier \* oth, Move \* mov )**

This function updates the status of each cell of the flip array for a move. If there is an alignment, the cell is set to 1, 0 otherwise.

**Parameters**

<i>oth</i>	: a pointer to the game board <a href="#">Othellier</a> structure
<i>mov</i>	: a pointer to a <a href="#">Move</a> structure action

**Returns**

1 if the array has been updated, 0 if the position of the move is a border (so the array has not been updated)

Tests whether the square designed by the position of the move is a border. If so, returns 0, if not, for each direction, tests if there may be an alignment. Calls the appropriate function for each direction ([has\\_alignment](#)), and sets the corresponding cell of the array to the return value of the function called. The function determines the validity of the alignment, or not.



#### 5.18.2.7 int update\_othellier ( **Game** \* *game* )

This function updates the status of the game board, and shows for each empty square if it is a legal move for the user who has the turn, or not.

## Parameters

<code>game</code>	: a pointer to the <a href="#">Game</a> structure
-------------------	---

## Returns

The number of legal squares in the new board

Allocates memory for a move. Sets all the previous LEGAL squares to EMPTY. Then, for each square, if it is empty, it sets the move on the position of the square and tests if the move is legal. If so, it puts the square to LEGAL and increments the counter. Then it reallocates the move and returns the counter.

## 5.19 othellier.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include "constants.h"
#include "structures.h"
```

### Functions

- [Othellier](#) \* [init\\_othellier](#) ()  
*This function initializes the main structure of the game board, so that one can begin a new game.*
- int [is\\_legal](#) ([Othellier](#) \*oth, [Move](#) \*mov)  
*This function determines whether the move passed in argument is a legal move on the game board passed in argument too.*
- int [update\\_flip](#) ([Othellier](#) \*oth, [Move](#) \*mov)  
*This function updates the status of each cell of the flip array for a move. If there is an alignment, the cell is set to 1, 0 otherwise.*
- short [has\\_alignment](#) ([Othellier](#) \*oth, short pos, short [color](#), short direction)  
*This function tests the validity and the existence of an alignment in the given direction (see direction Macros)*
- void [game\\_board\\_copy](#) ([Game](#) \*game)  
*This function copies the current game board state into the game board array (for saving)*
- int [destroy\\_end\\_boards\\_tab](#) ([Game](#) \*game)  
*This function frees the boards array from a cursor to another. It is used especially for freeing the end of the array, when a new move is played (after a cancellation).*
- int [update\\_othellier](#) ([Game](#) \*game)  
*This function updates the status of the game board, and shows for each empty square if it is a legal move for the user who has the turn, or not.*

### 5.19.1 Detailed Description

The header file of the [othellier.c](#) file. Here you can find the prototypes of all functions of this file

### 5.19.2 Function Documentation

#### 5.19.2.1 int [destroy\\_end\\_boards\\_tab](#) ( [Game](#) \* [game](#) )

This function frees the boards array from a cursor to another. It is used especially for freeing the end of the array, when a new move is played (after a cancellation).

## Parameters

<i>game</i>	: pointer to the <a href="#">Game</a> structure
-------------	---

## Returns

1 if at least 1 cell of the array has been freed, 0 otherwise

Tests if the cursors are equal or not. If not so, frees all cells of the array from the end to the cursor of the game. Reallocates a memory place for the new array (which is smaller).

## 5.19.2.2 void game\_board\_copy ( Game \* game )

This function copies the current game board state into the game board array (for saving)

## Parameters

<i>game</i>	: a pointer to the game structure
-------------	-----------------------------------

Allocates a bigger space to add the new board. Copies material, turn and each square of the board. Increments the game board cursor.

## 5.19.2.3 short has\_alignment ( Othellier \* oth, short pos, short color, short direction )

This function tests the validity and the existence of an alignment in the given direction (see direction Macros)

## Parameters

<i>oth</i>	: a pointer to the game board <a href="#">Othellier</a> structure
<i>pos</i>	: the position where the function needs to check for an alignment
<i>color</i>	: the color that the function needs to find at the end of the alignment to validate it
<i>direction</i>	: the direction of the alignment

## Returns

1 if there is a valid alignment, 0 otherwise

While the square in the given direction is the opponent's color, tests the next position in the same direction. If it's the needed color, returns 1, continues the loop otherwise, until the position has another value than the opponent's color.

## 5.19.2.4 Othellier\* init\_othellier ( )

This function initializes the main structure of the game board, so that one can begin a new game.

## Returns

A pointer to the game board [Othellier](#) structure, initialized

Allocates the game board and sets each square of the board to EMPTY, except the border, which are initialized to BORDER. Then the square 45 and 54 are transformed to BLACK, and the square 44 and 55 and transformed to WHITE. The pawn number of each player is set to 2, and the game board is returned.

This function initializes the main structure of the game board, so that one can begin a new game.

## 5.19.2.5 int is\_legal ( Othellier \* oth, Move \* mov )

This function determines whether the move passed in argument is a legal move on the game board passed in argument too.

**Parameters**

<i>oth</i>	a pointer to the game board
<i>mov</i>	: a pointer to a <a href="#">Move</a> action

**Returns**

YES if the move is legal, NO otherwise

The function tests if the move is legal by testing the value of the square on the game board. If the square is not empty, the move cannot be legal. Then, it tests for each cell of the flip array if there is any legal alignment

**5.19.2.6 int update\_flip ( Othellier \* oth, Move \* mov )**

This function updates the status of each cell of the flip array for a move. If there is an alignment, the cell is set to 1, 0 otherwise.

**Parameters**

<i>oth</i>	: a pointer to the game board <a href="#">Othellier</a> structure
<i>mov</i>	: a pointer to a <a href="#">Move</a> structure action

**Returns**

1 if the array has been updated, 0 if the position of the move is a border (so the array has not been updated)

Tests whether the square designed by the position of the move is a border. If so, returns 0, if not, for each direction, tests if there may be an alignment. Calls the appropriate function for each direction ([has\\_alignment](#)), and sets the corresponding cell of the array to the return value of the function called. The function determines the validity of the alignment, or not.

**5.19.2.7 int update\_othellier ( Game \* game )**

This function updates the status of the game board, and shows for each empty square if it is a legal move for the user who has the turn, or not.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

**Returns**

The number of legal squares in the new board

Allocates memory for a move. Sets all the previous LEGAL squares to EMPTY. Then, for each square, if it is empty, it sets the move on the position of the square and tests if the move is legal. If so, it puts the square to LEGAL and increments the counter. Then it reallocates the move and returns the counter.

**5.20 structures.h File Reference**

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

## Data Structures

- struct [Othellier](#)  
*Othellier* (game board) structure.
- struct [Player](#)  
*Player* structure.
- struct [Move](#)  
*Move* structure.
- struct [Game](#)  
*Game* structure.
- struct [Genome](#)  
*Genome* structure.
- struct [Population](#)  
*Population* structure.

### 5.20.1 Detailed Description

A file in which one can find all the data structures used in the project

## 5.21 user.c File Reference

```
#include "user.h"
```

## Functions

- short [choose\\_game\\_type](#) ()  
*This function allows to ask users for the type of game they want to play (Human vs Human or Human vs Computer)*
- short [choose\\_player\\_color](#) ()  
*This function allows to ask users for the color of pawns they want to have (for a Human vs Computer game)*
- void [enter\\_one\\_name](#) (Game \*game, short color)  
*This function allows to ask users for their player names.*
- void [enter\\_players\\_names](#) (Game \*game)  
*This function allows to ask users for their player names.*
- void [give\\_computer\\_name](#) (Game \*game, short color)  
*This function assigns a name to the computer player.*
- int [ask\\_save\\_game](#) (Game \*game)  
*This function allows to ask users whether they want to save the game in progress, and does it if they want to.*
- int [ask\\_new\\_game](#) ()  
*This function allows to ask users if they want to play the game again.*
- char \* [ask\\_load\\_game](#) ()  
*This function allows to ask users from what file they want to load the game.*
- void [display\\_usage](#) ()  
*This function displays a brief usage of the program.*
- void [display\\_help](#) ()  
*This function displays the help of the Othello program.*
- void [display\\_title](#) ()  
*This function displays the title of the program.*

### 5.21.1 Detailed Description

Functions related to the interactions between players and computer, including I/O interactions (such as asking players for an action)

### 5.21.2 Function Documentation

#### 5.21.2.1 `char* ask_load_game ( )`

This function allows to ask users from what file they want to load the game.

##### Returns

A string containing file name where the game needs to be loaded, NULL if the file doesn't exist or is not reachable

Asks for the name of the file, tests whether the file exists and tests if it can be opened in reading mode. Returns the name of the file.

#### 5.21.2.2 `int ask_new_game ( )`

This function allows to ask users if they want to play the game again.

##### Returns

YES if the answer is yes, NO otherwise

While users don't answer with a 'y' or a 'n', the function asks again. Returns the answer with YES or NO.

#### 5.21.2.3 `int ask_save_game ( Game * game )`

This function allows to ask users whether they want to save the game in progress, and does it if they want to.

##### Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

##### Returns

YES if the game has been saved, NO otherwise

Asks the players if want to save the game. If so, asks for a file name. Verifies that the file does not already exists. If it does, displays a warning and ask for cancellation. If not, it saves the game in the file specified and return YES if the file has been saved without errors.

#### 5.21.2.4 `short choose_game_type ( )`

This function allows to ask users for the type of game they want to play (Human vs Human or Human vs Computer)

##### Returns

1 (H\_VS\_H) if the choice is "Human vs Human" and 2 (H\_VS\_C) if the choice is "Human vs Computer"

While the player has not entered a correct response (1 or 2), the function asks the question again. Returns the response of the user .

## 5.21.2.5 short choose\_player\_color ( )

This function allows to ask users for the color of pawns they want to have (for a Human vs Computer game)

## Returns

0 for the color BLACK, 1 for the color WHITE

While the player has not entered a correct response (1 or 2), the function asks the question again. Returns the response of the user, minus 1 (to have the BLACK or WHITE code).

## 5.21.2.6 void display\_help ( )

This function displays the help of the Othello program.

The help contains the following options :

- N or NEW : Begins a new game
- L or LOAD : Loads a game from a text file
- S or SAVE : Saves the game in progress in a text file
- H or HELP : Displays the help
- C or CANCEL : Cancel the last move played
- U or UNCANCEL : Uncancel the last canceled move
- Q or QUIT : Exits the game

## 5.21.2.7 void display\_usage ( )

This function displays a brief usage of the program.

The usage contains :

- N or NE : Begins a new game
- H or HELP : Displays the help

## 5.21.2.8 void enter\_one\_name ( Game \* game, short color )

This function allows to ask users for their player names.

## Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color to which we want to give a name

While the player has not entered a correct name, the function asks the question again. When the player has entered his name, the function saves it in the [Game](#) structure in the reserved place. It verifies, if the color in argument is WHITE and the type of the black player is HUMAN, that the two names are different.

## 5.21.2.9 void enter\_players\_names ( Game \* game )

This function allows to ask users for their player names.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

Calls the function [enter\\_one\\_name\(\)](#) with two different parameters : BLACK for the black player and WHITE for the white one.

**5.21.2.10 void give\_computer\_name ( Game \* game, short color )**

This function assigns a name to the computer player.

**Parameters**

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color of the player the computer represents

Tests if the computer name is not already taken by the human player, and assigns it if not. Assigns an alternative name if so.

**5.22 user.h File Reference**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "constants.h"
#include "io.h"
#include "structures.h"
```

**Functions**

- short [choose\\_game\\_type](#) ()  
*This function allows to ask users for the type of game they want to play (Human vs Human or Human vs Computer)*
- short [choose\\_player\\_color](#) ()  
*This function allows to ask users for the color of pawns they want to have (for a Human vs Computer game)*
- void [enter\\_one\\_name](#) (Game \*game, short color)  
*This function allows to ask users for their player names.*
- void [enter\\_players\\_names](#) (Game \*game)  
*This function allows to ask users for their player names.*
- void [give\\_computer\\_name](#) (Game \*game, short color)  
*This function assigns a name to the computer player.*
- int [ask\\_save\\_game](#) (Game \*game)  
*This function allows to ask users whether they want to save the game in progress, and does it if they want to.*
- int [ask\\_new\\_game](#) ()  
*This function allows to ask users if they want to play the game again.*
- char \* [ask\\_load\\_game](#) ()  
*This function allows to ask users from what file they want to load the game.*
- void [display\\_usage](#) ()  
*This function displays a brief usage of the program.*
- void [display\\_help](#) ()  
*This function displays the help of the Othello program.*
- void [display\\_title](#) ()  
*This function displays the title of the program.*



### 5.22.1 Detailed Description

The header file of the [user.c](#) file. Here you can find the prototypes of all functions of this file

### 5.22.2 Function Documentation

#### 5.22.2.1 `char* ask_load_game ( )`

This function allows to ask users from what file they want to load the game.

##### Returns

A string containing file name where the game needs to be loaded, NULL if the file doesn't exist or is not reachable

Asks for the name of the file, tests whether the file exists and tests if it can be opened in reading mode. Returns the name of the file.

#### 5.22.2.2 `int ask_new_game ( )`

This function allows to ask users if they want to play the game again.

##### Returns

YES if the answer is yes, NO otherwise

While users don't answer with a 'y' or a 'n', the function asks again. Returns the answer with YES or NO.

#### 5.22.2.3 `int ask_save_game ( Game * game )`

This function allows to ask users whether they want to save the game in progress, and does it if they want to.

##### Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

##### Returns

YES if the game has been saved, NO otherwise

Asks the players if want to save the game. If so, asks for a file name. Verifies that the file does not already exist. If it does, displays a warning and ask for cancellation. If not, it saves the game in the file specified and return YES if the file has been saved without errors.

#### 5.22.2.4 `short choose_game_type ( )`

This function allows to ask users for the type of game they want to play (Human vs Human or Human vs Computer)

##### Returns

1 (H\_VS\_H) if the choice is "Human vs Human" and 2 (H\_VS\_C) if the choice is "Human vs Computer"

While the player has not entered a correct response (1 or 2), the function asks the question again. Returns the response of the user .

#### 5.22.2.5 short choose\_player\_color ( )

This function allows to ask users for the color of pawns they want to have (for a Human vs Computer game)

##### Returns

0 for the color BLACK, 1 for the color WHITE

While the player has not entered a correct response (1 or 2), the function asks the question again. Returns the response of the user, minus 1 (to have the BLACK or WHITE code).

#### 5.22.2.6 void display\_help ( )

This function displays the help of the Othello program.

The help contains the following options :

- N or NEW : Begins a new game
- L or LOAD : Loads a game from a text file
- S or SAVE : Saves the game in progress in a text file
- H or HELP : Displays the help
- C or CANCEL : Cancel the last move played
- U or UNCANCEL : Uncancel the last canceled move
- Q or QUIT : Exits the game

#### 5.22.2.7 void display\_usage ( )

This function displays a brief usage of the program.

The usage contains :

- N or NE : Begins a new game
- H or HELP : Displays the help

#### 5.22.2.8 void enter\_one\_name ( Game \* game, short color )

This function allows to ask users for their player names.

##### Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color to which we want to give a name

While the player has not entered a correct name, the function asks the question again. When the player has entered his name, the function saves it in the [Game](#) structure in the reserved place. It verifies, if the color in argument is WHITE and the type of the black player is HUMAN, that the two names are different.

#### 5.22.2.9 void enter\_players\_names ( Game \* game )

This function allows to ask users for their player names.

## Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
-------------	---

Calls the function [enter\\_one\\_name\(\)](#) with two different parameters : BLACK for the black player and WHITE for the white one.

5.22.2.10 void give\_computer\_name ( Game \* *game*, short *color* )

This function assigns a name to the computer player.

## Parameters

<i>game</i>	: a pointer to the <a href="#">Game</a> structure
<i>color</i>	: the color of the player the computer represents

Tests if the computer name is not already taken by the human player, and assigns it if not. Assigns an alternative name if so.

# Index

- ai.c, [13](#)
  - evaluate\_corner\_square\_pattern, [13](#)
  - evaluate\_long\_alignments, [14](#)
  - evaluation\_function, [14](#)
  - find\_legal\_moves, [14](#)
  - min\_max, [14](#)
  - number\_of\_corners, [16](#)
  - positions\_strength, [16](#)
- ai.h, [17](#)
  - evaluate\_corner\_square\_pattern, [17](#)
  - evaluate\_long\_alignments, [18](#)
  - evaluation\_function, [18](#)
  - find\_legal\_moves, [18](#)
  - min\_max, [18](#)
  - number\_of\_corners, [20](#)
  - positions\_strength, [20](#)
- are\_good\_coordinates
  - io.c, [45](#)
  - io.h, [51](#)
- array
  - Othellier, [10](#)
- ask\_YN
  - io.c, [45](#)
  - io.h, [51](#)
- ask\_load\_game
  - user.c, [68](#)
  - user.h, [71](#)
- ask\_new\_game
  - user.c, [68](#)
  - user.h, [71](#)
- ask\_save\_game
  - user.c, [68](#)
  - user.h, [71](#)
- boardsCursor
  - Game, [7](#)
- cancel\_last\_move
  - move.c, [57](#)
  - move.h, [59](#)
- char2int
  - io.c, [46](#)
  - io.h, [51](#)
- choose\_game\_type
  - user.c, [68](#)
  - user.h, [71](#)
- choose\_player\_color
  - user.c, [68](#)
  - user.h, [71](#)
- coefs
  - Game, [7](#)
  - Genome, [9](#)
- color
  - Move, [9](#)
- constants.h, [20](#)
  - GEN\_ALGO, [23](#)
- contains\_only\_letters
  - io.c, [46](#)
  - io.h, [52](#)
- create\_move
  - move.c, [57](#)
  - move.h, [59](#)
- cursor
  - Game, [7](#)
- destroy\_end\_boards\_tab
  - othellier.c, [61](#)
  - othellier.h, [64](#)
- destroy\_end\_moves\_tab
  - game.c, [30](#)
  - game.h, [33](#)
- destroy\_game
  - game.c, [30](#)
  - game.h, [34](#)
- destroy\_population
  - genetic.c, [38](#)
  - genetic.h, [41](#)
- display\_help
  - user.c, [69](#)
  - user.h, [72](#)
- display\_othellier
  - io.c, [46](#)
  - io.h, [52](#)
- display\_usage
  - user.c, [69](#)
  - user.h, [72](#)
- empty\_buffer
  - io.c, [46](#)
  - io.h, [52](#)
- end\_of\_game
  - game.c, [30](#)
  - game.h, [34](#)
- end\_of\_game\_action
  - game.c, [31](#)
  - game.h, [34](#)
- enter\_one\_name
  - user.c, [69](#)
  - user.h, [72](#)
- enter\_players\_names

- user.c, 69
- user.h, 72
- evaluate\_corner\_square\_pattern
  - ai.c, 13
  - ai.h, 17
- evaluate\_long\_alignments
  - ai.c, 14
  - ai.h, 18
- evaluation\_function
  - ai.c, 14
  - ai.h, 18
- event\_cancel
  - events.c, 24
  - events.h, 28
- event\_help
  - events.c, 24
  - events.h, 28
- event\_load
  - events.c, 24
  - events.h, 28
- event\_new
  - events.c, 25
  - events.h, 28
- event\_other
  - events.c, 25
  - events.h, 28
- event\_quit
  - events.c, 25
  - events.h, 29
- event\_save
  - events.c, 25
  - events.h, 29
- event\_uncancel
  - events.c, 27
  - events.h, 29
- events.c, 23
  - event\_cancel, 24
  - event\_help, 24
  - event\_load, 24
  - event\_new, 25
  - event\_other, 25
  - event\_quit, 25
  - event\_save, 25
  - event\_uncancel, 27
- events.h, 27
  - event\_cancel, 28
  - event\_help, 28
  - event\_load, 28
  - event\_new, 28
  - event\_other, 28
  - event\_quit, 29
  - event\_save, 29
  - event\_uncancel, 29
- exists\_file
  - io.c, 46
  - io.h, 52
- find\_legal\_moves
  - ai.c, 14
- ai.h, 18
- fitness
  - genetic.c, 38
  - genetic.h, 41
  - Genome, 9
- flip
  - Move, 9
- GEN\_ALGO
  - constants.h, 23
- Game, 7
  - boardsCursor, 7
  - coefs, 7
  - cursor, 7
  - moves, 7
  - movesCursor, 8
  - oth, 8
  - oths, 8
  - p1, 8
  - p2, 8
  - saved, 8
  - turn, 8
- game.c, 29
  - destroy\_end\_moves\_tab, 30
  - destroy\_game, 30
  - end\_of\_game, 30
  - end\_of\_game\_action, 31
  - has\_legal\_move, 31
  - init\_game, 31
  - is\_computer\_game, 31
  - load\_coefs, 32
  - play\_ai, 32
  - update\_turn, 32
  - whose\_turn\_is\_it, 32
- game.h, 33
  - destroy\_end\_moves\_tab, 33
  - destroy\_game, 34
  - end\_of\_game, 34
  - end\_of\_game\_action, 34
  - has\_legal\_move, 34
  - init\_game, 35
  - is\_computer\_game, 35
  - load\_coefs, 35
  - play\_ai, 35
  - update\_turn, 35
  - whose\_turn\_is\_it, 37
- game\_board\_copy
  - othellier.c, 61
  - othellier.h, 65
- game\_saving
  - io.c, 48
  - io.h, 52
- genNb
  - Population, 11
- genetic.c, 37
  - destroy\_population, 38
  - fitness, 38
  - genetic\_algorithm, 38
  - initial\_population, 38

- mutation, 38
  - next\_generation, 39
  - random\_tenth, 39
  - reproduction, 39
  - save\_coefs, 39
  - search\_best\_genome, 40
- genetic.h, 40
  - destroy\_population, 41
  - fitness, 41
  - genetic\_algorithm, 41
  - initial\_population, 41
  - mutation, 41
  - next\_generation, 42
  - random\_tenth, 42
  - reproduction, 42
  - save\_coefs, 42
  - search\_best\_genome, 43
- genetic\_algorithm
  - genetic.c, 38
  - genetic.h, 41
- Genome, 8
  - coefs, 9
  - fitness, 9
- genomes
  - Population, 11
- give\_computer\_name
  - user.c, 70
  - user.h, 73
- graphics.c, 43
- graphics.h, 43
- has\_alignment
  - othellier.c, 61
  - othellier.h, 65
- has\_legal\_move
  - game.c, 31
  - game.h, 34
- init\_game
  - game.c, 31
  - game.h, 35
- init\_othellier
  - othellier.c, 62
  - othellier.h, 65
- initial\_population
  - genetic.c, 38
  - genetic.h, 41
- int2str
  - io.c, 48
  - io.h, 54
- io.c, 44
  - are\_good\_coordinates, 45
  - ask\_YN, 45
  - char2int, 46
  - contains\_only\_letters, 46
  - display\_othellier, 46
  - empty\_buffer, 46
  - exists\_file, 46
  - game\_saving, 48
  - int2str, 48
  - is\_a\_letter, 48
  - load\_game, 48
  - str2upper, 50
- io.h, 50
  - are\_good\_coordinates, 51
  - ask\_YN, 51
  - char2int, 51
  - contains\_only\_letters, 52
  - display\_othellier, 52
  - empty\_buffer, 52
  - exists\_file, 52
  - game\_saving, 52
  - int2str, 54
  - is\_a\_letter, 54
  - load\_game, 54
  - str2upper, 54
- is\_a\_letter
  - io.c, 48
  - io.h, 54
- is\_computer\_game
  - game.c, 31
  - game.h, 35
- is\_legal
  - othellier.c, 62
  - othellier.h, 65
- load\_coefs
  - game.c, 32
  - game.h, 35
- load\_game
  - io.c, 48
  - io.h, 54
- main
  - main.c, 55
  - main.h, 56
- main.c, 55
  - main, 55
- main.h, 55
  - main, 56
- min\_max
  - ai.c, 14
  - ai.h, 18
- Move, 9
  - color, 9
  - flip, 9
  - position, 9
- move.c, 56
  - cancel\_last\_move, 57
  - create\_move, 57
  - move\_processing, 57
  - play\_move, 58
  - uncancel\_move, 58
- move.h, 58
  - cancel\_last\_move, 59
  - create\_move, 59
  - move\_processing, 59
  - play\_move, 60

- uncancel\_move, 60
- move\_processing
  - move.c, 57
  - move.h, 59
- moves
  - Game, 7
- movesCursor
  - Game, 8
- mutation
  - genetic.c, 38
  - genetic.h, 41
- next\_generation
  - genetic.c, 39
  - genetic.h, 42
- number\_of\_corners
  - ai.c, 16
  - ai.h, 20
- oth
  - Game, 8
- Othellier, 9
  - array, 10
  - piecesNb, 10
  - turn, 10
- othellier.c, 60
  - destroy\_end\_boards\_tab, 61
  - game\_board\_copy, 61
  - has\_alignment, 61
  - init\_othellier, 62
  - is\_legal, 62
  - update\_flip, 62
  - update\_othellier, 62
- othellier.h, 64
  - destroy\_end\_boards\_tab, 64
  - game\_board\_copy, 65
  - has\_alignment, 65
  - init\_othellier, 65
  - is\_legal, 65
  - update\_flip, 66
  - update\_othellier, 66
- oths
  - Game, 8
- p1
  - Game, 8
- p2
  - Game, 8
- piecesNb
  - Othellier, 10
- play\_ai
  - game.c, 32
  - game.h, 35
- play\_move
  - move.c, 58
  - move.h, 60
- Player, 10
  - player\_name, 10
  - player\_type, 10
- player\_name
  - Player, 10
- player\_type
  - Player, 10
- Population, 11
  - genNb, 11
  - genomes, 11
- position
  - Move, 9
- positions\_strength
  - ai.c, 16
  - ai.h, 20
- random\_tenth
  - genetic.c, 39
  - genetic.h, 42
- reproduction
  - genetic.c, 39
  - genetic.h, 42
- save\_coefs
  - genetic.c, 39
  - genetic.h, 42
- saved
  - Game, 8
- search\_best\_genome
  - genetic.c, 40
  - genetic.h, 43
- str2upper
  - io.c, 50
  - io.h, 54
- structures.h, 66
- turn
  - Game, 8
  - Othellier, 10
- uncancel\_move
  - move.c, 58
  - move.h, 60
- update\_flip
  - othellier.c, 62
  - othellier.h, 66
- update\_othellier
  - othellier.c, 62
  - othellier.h, 66
- update\_turn
  - game.c, 32
  - game.h, 35
- user.c, 67
  - ask\_load\_game, 68
  - ask\_new\_game, 68
  - ask\_save\_game, 68
  - choose\_game\_type, 68
  - choose\_player\_color, 68
  - display\_help, 69
  - display\_usage, 69
  - enter\_one\_name, 69
  - enter\_players\_names, 69

- give\_computer\_name, [70](#)
- user.h, [70](#)
  - ask\_load\_game, [71](#)
  - ask\_new\_game, [71](#)
  - ask\_save\_game, [71](#)
  - choose\_game\_type, [71](#)
  - choose\_player\_color, [71](#)
  - display\_help, [72](#)
  - display\_usage, [72](#)
  - enter\_one\_name, [72](#)
  - enter\_players\_names, [72](#)
  - give\_computer\_name, [73](#)
- whose\_turn\_is\_it
  - game.c, [32](#)
  - game.h, [37](#)