

## Four-in-a-Row

Generated by Doxygen 1.8.17

---

<b>1 Four-in-a-row</b>	<b>1</b>
<b>2 TODO</b>	<b>1</b>
<b>3 Hierarchical Index</b>	<b>1</b>
3.1 Class Hierarchy . . . . .	1
<b>4 Data Structure Index</b>	<b>1</b>
4.1 Data Structures . . . . .	1
<b>5 File Index</b>	<b>2</b>
5.1 File List . . . . .	2
<b>6 Data Structure Documentation</b>	<b>3</b>
6.1 ClientSocketWrapper Class Reference . . . . .	3
6.1.1 Detailed Description . . . . .	3
6.1.2 Member Function Documentation . . . . .	3
6.2 Connect4 Class Reference . . . . .	4
6.2.1 Detailed Description . . . . .	4
6.2.2 Constructor & Destructor Documentation . . . . .	4
6.2.3 Member Function Documentation . . . . .	5
6.3 Host Class Reference . . . . .	7
6.3.1 Detailed Description . . . . .	7
6.3.2 Constructor & Destructor Documentation . . . . .	7
6.4 Message Class Reference . . . . .	8
6.4.1 Detailed Description . . . . .	8
6.4.2 Member Function Documentation . . . . .	9
6.5 MoveMessage Class Reference . . . . .	9
6.5.1 Detailed Description . . . . .	10
6.5.2 Member Function Documentation . . . . .	10
6.6 ServerSocketWrapper Class Reference . . . . .	10
6.6.1 Detailed Description . . . . .	11
6.6.2 Constructor & Destructor Documentation . . . . .	11
6.7 SocketWrapper Class Reference . . . . .	12
6.7.1 Detailed Description . . . . .	12
6.7.2 Member Function Documentation . . . . .	12
6.8 StartGameMessage Class Reference . . . . .	14
6.8.1 Detailed Description . . . . .	15
6.8.2 Member Function Documentation . . . . .	15
<b>7 File Documentation</b>	<b>15</b>
7.1 client.cpp File Reference . . . . .	15
7.1.1 Detailed Description . . . . .	16
7.2 client.cpp . . . . .	16

7.3 connect4.cpp File Reference . . . . .	18
7.3.1 Detailed Description . . . . .	19
7.4 connect4.cpp . . . . .	19
7.5 connect4.h File Reference . . . . .	21
7.5.1 Detailed Description . . . . .	21
7.6 connect4.h . . . . .	22
7.7 dump_buffer.cpp File Reference . . . . .	22
7.7.1 Detailed Description . . . . .	22
7.7.2 Function Documentation . . . . .	23
7.8 dump_buffer.cpp . . . . .	23
7.9 dump_buffer.h File Reference . . . . .	23
7.9.1 Detailed Description . . . . .	24
7.9.2 Function Documentation . . . . .	24
7.10 dump_buffer.h . . . . .	24
7.11 host.h File Reference . . . . .	25
7.11.1 Detailed Description . . . . .	25
7.12 host.h . . . . .	25
7.13 inet_utils.cpp File Reference . . . . .	26
7.13.1 Detailed Description . . . . .	26
7.13.2 Function Documentation . . . . .	26
7.14 inet_utils.cpp . . . . .	28
7.15 inet_utils.h File Reference . . . . .	29
7.15.1 Detailed Description . . . . .	30
7.15.2 Function Documentation . . . . .	30
7.16 inet_utils.h . . . . .	32
7.17 logging.h File Reference . . . . .	33
7.17.1 Detailed Description . . . . .	33
7.18 logging.h . . . . .	34
7.19 messages.cpp File Reference . . . . .	35
7.19.1 Detailed Description . . . . .	35
7.19.2 Function Documentation . . . . .	35
7.20 messages.cpp . . . . .	36
7.21 messages.h File Reference . . . . .	36
7.21.1 Detailed Description . . . . .	37
7.21.2 Macro Definition Documentation . . . . .	37
7.21.3 Enumeration Type Documentation . . . . .	38
7.21.4 Function Documentation . . . . .	38
7.22 messages.h . . . . .	38
7.23 single_player.cpp File Reference . . . . .	39
7.23.1 Detailed Description . . . . .	39
7.24 single_player.cpp . . . . .	40
7.25 socket_wrapper.cpp File Reference . . . . .	41

7.25.1 Detailed Description . . . . .	41
7.26 socket_wrapper.cpp . . . . .	41
7.27 socket_wrapper.h File Reference . . . . .	43
7.27.1 Detailed Description . . . . .	43
7.28 socket_wrapper.h . . . . .	44
<b>Index</b>	<b>45</b>

## 1 Four-in-a-row

Four-in-a-row game for the course of Foundations of Cyber Security at University of Pisa

## 2 TODO

Check for full board when we play without winning

## 3 Hierarchical Index

### 3.1 Class Hierarchy

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

<b>Connect4</b>	<b>4</b>
<b>Host</b>	<b>7</b>
<b>Message</b>	<b>8</b>
<b>MoveMessage</b>	<b>9</b>
<b>StartGameMessage</b>	<b>14</b>
<b>SocketWrapper</b>	<b>12</b>
<b>ClientSocketWrapper</b>	<b>3</b>
<b>ServerSocketWrapper</b>	<b>10</b>

## 4 Data Structure Index

### 4.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">ClientSocketWrapper</a> <a href="#">SocketWrapper</a> for a TCP client	3
<a href="#">Connect4</a>	4
<a href="#">Host</a> Class that holds a host information	7
<a href="#">Message</a> Abstract class for Messages	8
<a href="#">MoveMessage</a> <a href="#">Message</a> that signals a move	9
<a href="#">ServerSocketWrapper</a> <a href="#">SocketWrapper</a> for a TCP server	10
<a href="#">SocketWrapper</a> Wrapper class around sockaddr_in and socket descriptor	12
<a href="#">StartGameMessage</a> <a href="#">Message</a> that signals to start a new game	14

## 5 File Index

### 5.1 File List

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

<a href="#">client.cpp</a> Implementation of a 4-in-a-row game	15
<a href="#">connect4.cpp</a> Implementation of <a href="#">connect4.h</a>	18
<a href="#">connect4.h</a> Header file for the class responsible of handling the board of a <a href="#">Connect4</a> game	21
<a href="#">dump_buffer.cpp</a> Implementation of <a href="#">dump_buffer.h</a>	22
<a href="#">dump_buffer.h</a> Utility function for dumping a buffer as hex string	23
<a href="#">host.h</a> Definition of the helper class "Host"	25
<a href="#">inet_utils.cpp</a> Implementation of <a href="#">inet_utils.h</a>	26
<a href="#">inet_utils.h</a> Utility functions for managing inet addresses	29
<a href="#">logging.h</a> Logging macro	33
<a href="#">messages.cpp</a> Implementation of <a href="#">messages.h</a>	35

<a href="#">messages.h</a>	
Definition of messages	36
<a href="#">single_player.cpp</a>	
Implementation of a 4-in-a-row game	39
<a href="#">socket_wrapper.cpp</a>	
Implementation of <a href="#">socket_wrapper.h</a>	41
<a href="#">socket_wrapper.h</a>	
Definition of the helper class "SocketWrapper" and derivatives	43

## 6 Data Structure Documentation

### 6.1 ClientSocketWrapper Class Reference

[SocketWrapper](#) for a TCP client.

```
#include <socket_wrapper.h>
```

Inherits [SocketWrapper](#).

#### Public Member Functions

- int [connectServer](#) ([Host](#) host)  
*Connects to a remote server.*

#### Additional Inherited Members

#### 6.1.1 Detailed Description

[SocketWrapper](#) for a TCP client.

It provides a new function to connect to server.

Definition at line 110 of file [socket\\_wrapper.h](#).

#### 6.1.2 Member Function Documentation

**6.1.2.1 connectServer()** int ClientSocketWrapper::connectServer (  
[Host](#) host )

Connects to a remote server.

##### Returns

0 in case of success, something else otherwise

Definition at line 85 of file [socket\\_wrapper.cpp](#).

## 6.2 Connect4 Class Reference

### Public Member Functions

- [Connect4](#) (int rows=6, int columns=7)  
*Construct a new Connect 4 object.*
- int [getNumCols](#) ()  
*Get the number of columns of the board.*
- int8\_t [play](#) (int column, char player=0)  
*Inserts a token.*
- bool [checkWin](#) (int starting\_row, int starting\_col, char player=0)  
*Checks if an inserted token causes a win.*
- bool [setPlayer](#) (char player)  
*Sets the default player.*
- char [getPlayer](#) ()  
*Get the default player.*
- char [getAdv](#) ()  
*Get the adversary, when a default player is set.*
- void [print](#) (std::ostream &os)  
*Prints the board.*

### Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [Connect4](#) &b)

### 6.2.1 Detailed Description

Definition at line 18 of file [connect4.h](#).

### 6.2.2 Constructor & Destructor Documentation

**6.2.2.1 [Connect4\(\)](#)** `Connect4::Connect4 (`  
    `int rows = 6,`  
    `int columns = 7 )`

Construct a new Connect 4 object.

#### Parameters

<i>rows</i>	Number of rows
<i>columns</i>	Number of columns

Definition at line 19 of file [connect4.cpp](#).

### 6.2.3 Member Function Documentation

**6.2.3.1 checkWin()** `bool Connect4::checkWin (`  
    `int starting_row,`  
    `int starting_col,`  
    `char player = 0 )`

Checks if an inserted token causes a win.

#### Parameters

<i>starting_row</i>	row of the token
<i>starting_col</i>	col of the token
<i>player</i>	marker of the player inserting the token

#### Returns

true if winning, false otherwise

Definition at line 83 of file [connect4.cpp](#).

**6.2.3.2 getAdv()** `char Connect4::getAdv ( )`

Get the adversary, when a default player is set.

#### Returns

enemy marker

Definition at line 146 of file [connect4.cpp](#).

**6.2.3.3 getNumCols()** `int Connect4::getNumCols ( )`

Get the number of columns of the board.

#### Returns

number of columns

Definition at line 128 of file [connect4.cpp](#).



**6.2.3.4 getPlayer()** `char Connect4::getPlayer ( )`

Get the default player.

**Returns**

player marker

Definition at line 142 of file [connect4.cpp](#).

**6.2.3.5 play()** `int8_t Connect4::play (`  
`int column,`  
`char player = 0 )`

Inserts a token.

**Parameters**

<i>column</i>	target column where the token should be added
<i>player</i>	player who is making the move

**Return values**

1	Success with win
0	Success without win
-1	Failure for full column
-2	Board is full, it could be so before or after the move takes place

Definition at line 31 of file [connect4.cpp](#).

**6.2.3.6 print()** `void Connect4::print (`  
`std::ostream & os )`

Prints the board.

**Parameters**

<i>os</i>	Output stream where the board has to be printed
-----------	---

Definition at line 15 of file [connect4.cpp](#).

**6.2.3.7 setPlayer()** `bool Connect4::setPlayer (`  
`char player )`

Sets the default player.

## Parameters

<i>player</i>	player to be set
---------------	------------------

## Returns

true if a valid player was supplied and set  
false otherwise

Definition at line 132 of file [connect4.cpp](#).

## 6.3 Host Class Reference

Class that holds a host information.

```
#include <host.h>
```

### Public Member Functions

- [Host](#) (struct sockaddr\_in addr)  
*Constructs new instance from given inet address.*
- [Host](#) (char \*ip, int port)  
*Constructs new instance from IP/port pair.*
- struct sockaddr\_in [getAddress](#) ()  
*Returns the inet address of the host.*
- string [toString](#) ()  
*Returns the inet address of the host.*

#### 6.3.1 Detailed Description

Class that holds a host information.

At the moment, it only holds its inet addr but in the future its public key and other OpenSSL stuff will be put here.

Definition at line 23 of file [host.h](#).

#### 6.3.2 Constructor & Destructor Documentation

**6.3.2.1 Host() [1/2]** Host::Host (  
struct sockaddr\_in addr ) [inline]

Constructs new instance from given inet address.

**Parameters**

<i>addr</i>	the inet address of the remote host
-------------	-------------------------------------

Definition at line 33 of file [host.h](#).

**6.3.2.2 Host()** [2/2] `Host::Host (`  
    `char * ip,`  
    `int port ) [inline]`

Constructs new instance from IP/port pair.

**Parameters**

<i>addr</i>	the inet address of the remote host
-------------	-------------------------------------

Definition at line 41 of file [host.h](#).

## 6.4 Message Class Reference

Abstract class for Messages.

```
#include <messages.h>
```

Inherited by [MoveMessage](#), and [StartGameMessage](#).

**Public Member Functions**

- virtual int [write](#) (char \*buffer)=0  
    *Write message to buffer.*
- virtual int [read](#) (char \*buffer, int len)=0  
    *Read message from buffer.*
- virtual size\_t [size](#) ()=0  
    *Get required buffer size.*
- virtual string [getName](#) ()=0  
    *Get message name (for debug purposes)*
- virtual [MessageType](#) [getType](#) ()=0

### 6.4.1 Detailed Description

Abstract class for Messages.

Definition at line 37 of file [messages.h](#).

## 6.4.2 Member Function Documentation

**6.4.2.1 read()** `virtual int Message::read (`  
    `char * buffer,`  
    `int len ) [pure virtual]`

Read message from buffer.

### Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implemented in [MoveMessage](#), and [StartGameMessage](#).

**6.4.2.2 write()** `virtual int Message::write (`  
    `char * buffer ) [pure virtual]`

Write message to buffer.

### Returns

number of bytes written

Implemented in [MoveMessage](#), and [StartGameMessage](#).

## 6.5 MoveMessage Class Reference

[Message](#) that signals a move.

```
#include <messages.h>
```

Inherits [Message](#).

### Public Member Functions

- **MoveMessage** (char col)
- int [write](#) (char \*buffer)  
    *Write message to buffer.*
- int [read](#) (char \*buffer, int len)  
    *Read message from buffer.*
- size\_t [size](#) ()  
    *Get required buffer size.*
- string [getName](#) ()  
    *Get message name (for debug purposes)*
- char **getColumn** ()
- [MessageType](#) **getType** ()

### 6.5.1 Detailed Description

[Message](#) that signals a move.

Definition at line 87 of file [messages.h](#).

### 6.5.2 Member Function Documentation

**6.5.2.1 read()** `int MoveMessage::read (
 char * buffer,
 int len ) [virtual]`

Read message from buffer.

#### Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements [Message](#).

Definition at line 60 of file [messages.cpp](#).

**6.5.2.2 write()** `int MoveMessage::write (
 char * buffer ) [virtual]`

Write message to buffer.

#### Returns

number of bytes written

Implements [Message](#).

Definition at line 54 of file [messages.cpp](#).

## 6.6 ServerSocketWrapper Class Reference

[SocketWrapper](#) for a TCP server.

```
#include <socket_wrapper.h>
```

Inherits [SocketWrapper](#).

## Public Member Functions

- [ServerSocketWrapper](#) ()  
*Initialize a new socket on a random port.*
- [ServerSocketWrapper](#) (int port)  
*Initialize a new socket at the requested port.*
- [SocketWrapper](#) \* [acceptClient](#) ()  
*Accepts any incoming connection and returns the related [SocketWrapper](#).*
- int [getPort](#) ()  
*Returns port the server is listening new connections on.*

## Additional Inherited Members

### 6.6.1 Detailed Description

[SocketWrapper](#) for a TCP server.

It provides a new function to accept clients. Constructor also set listen mode.

Definition at line 126 of file [socket\\_wrapper.h](#).

### 6.6.2 Constructor & Destructor Documentation

#### 6.6.2.1 [ServerSocketWrapper](#)() [1/2] `ServerSocketWrapper::ServerSocketWrapper ( )`

Initialize a new socket on a random port.

##### Parameters

<i>port</i>	the port you want to bind on
-------------	------------------------------

Definition at line 106 of file [socket\\_wrapper.cpp](#).

#### 6.6.2.2 [ServerSocketWrapper](#)() [2/2] `ServerSocketWrapper::ServerSocketWrapper (int port )`

Initialize a new socket at the requested port.

##### Parameters

<i>port</i>	the port you want to bind on
-------------	------------------------------

Definition at line 121 of file [socket\\_wrapper.cpp](#).

## 6.7 SocketWrapper Class Reference

Wrapper class around sockaddr\_in and socket descriptor.

```
#include <socket_wrapper.h>
```

Inherited by [ClientSocketWrapper](#), and [ServerSocketWrapper](#).

### Public Member Functions

- [SocketWrapper](#) ()  
*Initialize on a new socket.*
- [SocketWrapper](#) (int sd)  
*Initialize using existing socket.*
- int [getDescriptor](#) ()  
*Returns current socket file descriptor.*
- [Message](#) \* [receiveAnyMsg](#) (size\_t size=[MAX\\_MSG\\_SIZE](#))  
*Receive any new message from the socket.*
- [Message](#) \* [receiveMsg](#) ([MessageType](#) type, size\_t size=[MAX\\_MSG\\_SIZE](#))  
*Receive a new message of the given type from the socket.*
- [Message](#) \* [receiveMsg](#) ([MessageType](#) type[], int n\_types, size\_t size=[MAX\\_MSG\\_SIZE](#))  
*Receive a new message of any of the given types from the socket.*
- int [sendMsg](#) ([Message](#) \*msg)  
*Sends the given message to the peer host through the socket.*
- void [setOtherAddr](#) (struct sockaddr\_in addr)  
*Sets the address of the other host.*
- [Host](#) [getConnectedHost](#) ()  
*Returns connected host.*

### Protected Attributes

- struct sockaddr\_in [other\\_addr](#)  
*Other host inet socket.*
- int [socket\\_fd](#)  
*Socket file descriptor.*

#### 6.7.1 Detailed Description

Wrapper class around sockaddr\_in and socket descriptor.

It provides a more simple interface saving a lot of boiler-plate code. There are two subclasses: [ClientSocketWrapper](#) and [ServerSocketWrapper](#).

Definition at line 24 of file [socket\\_wrapper.h](#).

#### 6.7.2 Member Function Documentation

**6.7.2.1 receiveAnyMsg()** [Message](#) \* [SocketWrapper::receiveAnyMsg](#) (  
size\_t size = [MAX\\_MSG\\_SIZE](#) )

Receive any new message from the socket.

This API is blocking.

## Parameters

<i>size</i>	the size of the temporary buffer
-------------	----------------------------------

## Returns

the received message or null if an error occurred

Definition at line 22 of file [socket\\_wrapper.cpp](#).

**6.7.2.2 receiveMsg()** [1/2] `Message * SocketWrapper::receiveMsg (`  
    `MessageType type,`  
    `size_t size = MAX_MSG_SIZE )`

Receive a new message of the given type from the socket.

When a message of the wrong type is received it is simply ignored.

This API is blocking.

## Parameters

<i>type</i>	the type to keep
<i>size</i>	the size of the temporary buffer

## Returns

the received message or null if an error occurred

Definition at line 37 of file [socket\\_wrapper.cpp](#).

**6.7.2.3 receiveMsg()** [2/2] `Message * SocketWrapper::receiveMsg (`  
    `MessageType type[],`  
    `int n_types,`  
    `size_t size = MAX_MSG_SIZE )`

Receive a new message of any of the given types from the socket.

When a message of the wrong type is received it is simply ignored.

This API is blocking.

## Parameters

<i>type</i>	the types to keep (array)
<i>n_types</i>	the number of types to keep (array length)



**Returns**

the received message or null if an error occurred

Definition at line 41 of file [socket\\_wrapper.cpp](#).

**6.7.2.4 sendMsg()** `int SocketWrapper::sendMsg ( Message * msg )`

Sends the given message to the peer host through the socket.

**Parameters**

<code>msg</code>	the message to be sent
------------------	------------------------

**Returns**

0 in case of success, something else otherwise

Definition at line 60 of file [socket\\_wrapper.cpp](#).

**6.7.2.5 setOtherAddr()** `void SocketWrapper::setOtherAddr ( struct sockaddr_in addr ) [inline]`

Sets the address of the other host.

This is used when initializing a new [SocketWrapper](#) for a newly acceptor connection.

Definition at line 97 of file [socket\\_wrapper.h](#).

## 6.8 StartGameMessage Class Reference

[Message](#) that signals to start a new game.

```
#include <messages.h>
```

Inherits [Message](#).

**Public Member Functions**

- `int write (char *buffer)`  
*Write message to buffer.*
- `int read (char *buffer, int len)`  
*Read message from buffer.*
- `size_t size ()`  
*Get required buffer size.*
- `string getName ()`  
*Get message name (for debug purposes)*
- `MessageType getType ()`

### 6.8.1 Detailed Description

[Message](#) that signals to start a new game.

Definition at line 70 of file [messages.h](#).

### 6.8.2 Member Function Documentation

**6.8.2.1 read()** `int StartGameMessage::read (
 char * buffer,
 int len ) [virtual]`

Read message from buffer.

#### Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements [Message](#).

Definition at line 50 of file [messages.cpp](#).

**6.8.2.2 write()** `int StartGameMessage::write (
 char * buffer ) [virtual]`

Write message to buffer.

#### Returns

number of bytes written

Implements [Message](#).

Definition at line 45 of file [messages.cpp](#).

## 7 File Documentation

### 7.1 client.cpp File Reference

Implementation of a 4-in-a-row game.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "connect4.h"
#include "logging.h"
#include "network/socket_wrapper.h"
#include "network/host.h"
```

- ## Functions

- ## Variables

- Author**

Date \_\_\_\_\_

Definition in file [client.cpp](#).

## 7.2 client.cpp

Generated by Doxygen

```

00040         <<"*****"
00041         <<endl;
00042     }
00043
00044     int main(int argc, char** argv){
00045         // handle server selection on params here
00046         char in_buffer[256];
00047         int choosen_col;
00048         int adv_col;
00049         int win;
00050         int turn;
00051         SocketWrapper *sw;
00052         Host* peer;
00053         srand(time(NULL));
00054
00055         printWelcome();
00056         cout<<endl<<"Welcome to 4-in-a-row!"<<endl;
00057         cout<<"The rules of the game are simple: you win when you have 4 connected tokens along any
direction."<<endl;
00058
00059         Connect4 c;
00060         cout<<"Who do you want to be? X or O ?"<<endl;
00061
00062         do {
00063             cout<<"> ";
00064             cin.getline(in_buffer, sizeof(in_buffer));
00065         } while (!c.setPlayer(in_buffer[0]));
00066         cout<<"You are playing as " <<c.getPlayer()<<endl;
00067
00068
00069         if (argc != 1 && argc != 2 && argc != 3){
00070             print_help();
00071             return 1;
00072         } else if (argc == 1 || argc == 2){ // wait for peer connection
00073             ServerSocketWrapper *ssw;
00074             if (argc == 1){
00075                 ssw = new ServerSocketWrapper();
00076             } else{
00077                 ssw = new ServerSocketWrapper(atoi(argv[1]));
00078             }
00079             cout<<"Waiting for connection on port: " <<ssw->getPort()<<endl;
00080
00081             sw = ssw->acceptClient();
00082
00083             Host p = sw->getConnectedHost();
00084             peer = &p;
00085
00086             cout<<"Accepted client: " <<peer->toString()<<endl;
00087
00088             StartGameMessage *sgm = dynamic_cast<StartGameMessage*>(sw->receiveMsg(START_GAME));
00089             if (sgm == NULL){
00090                 LOG(LOG_ERR, "Connection error");
00091                 return 1;
00092             }
00093
00094             LOG(LOG_INFO, "Connected to %s", peer->toString().c_str());
00095             turn = MY_TURN;
00096         } else if (argc == 3){
00097             peer = new Host(argv[1], atoi(argv[2]));
00098
00099             cout<<"Connecting to: " <<peer->toString()<<endl;
00100
00101             ClientSocketWrapper *csw = new ClientSocketWrapper();
00102
00103             int ret = csw->connectServer(*peer);
00104
00105             if (ret != 0){
00106                 LOG(LOG_ERR, "Connection error");
00107                 return 1;
00108             }
00109
00110             sw = csw;
00111
00112             StartGameMessage m;
00113             ret = sw->sendMsg(&m);
00114
00115             if (ret != 0){
00116                 LOG(LOG_ERR, "Connection error");
00117                 return 1;
00118             }
00119
00120             LOG(LOG_INFO, "Connected to %s", peer->toString().c_str());
00121             turn = THEIR_TURN;
00122         }
00123
00124         cout<<"This is the starting board:"<<endl;
00125         cout<<c;

```

```

00126
00127     do {
00128         if (turn == MY_TURN){
00129             cout<<"Write the column you want to insert the token to"<<endl;
00130             do {
00131                 cout<<"> ";
00132                 cin.getline(in_buffer, sizeof(in_buffer));
00133                 choosen_col = in_buffer[0] - '0';
00134             } while(choosen_col < 0 || choosen_col > 7);
00135
00136             win = c.play(choosen_col-1, c.getPlayer());
00137             cout<<c;
00138
00139             if (win != -1){
00140                 MoveMessage mm(choosen_col-1);
00141                 int ret = sw->sendMsg(&mm);
00142                 if (ret != 0){
00143                     LOG(LOG_ERR, "Connection error");
00144                     return 1;
00145                 }
00146             }
00147
00148             if(win == 1){
00149                 cout<<"Congratulation, you won!"<<endl;
00150             } else if(win == -1){
00151                 cout<<"The column is full, choose a different one!"<<endl;
00152                 continue;
00153             } else if(win == -2){
00154                 cout<<"The entire board is filled: it is a draw!"<<endl;
00155                 break;
00156             } else{
00157                 turn = THEIR_TURN;
00158             }
00159
00160         } else{           // THEIR_TURN
00161             do {
00162                 MoveMessage *mm;
00163                 mm = dynamic_cast<MoveMessage*>(sw->receiveMsg(MOVE));
00164                 if (mm == NULL){
00165                     LOG(LOG_ERR, "Connection error");
00166                     return 1;
00167                 }
00168                 adv_col = mm->getColumn();
00169                 cout<<"Your enemy has chosen column "<<adv_col<<endl;
00170                 win = c.play(adv_col, c.getAdv());
00171                 cout<<c;
00172                 if(win == 1){
00173                     cout<<"Damn! You lost!"<<endl;
00174                 } else if(win == -1){
00175                     cout<<"The column is full, the adversary has lost!"<<endl;
00176                     break;
00177                 } else if(win == -2){
00178                     cout<<"The entire board is filled: it is a draw!"<<endl;
00179                     break;
00180                 }
00181             } while (win == -1);
00182             turn = MY_TURN;
00183         }
00184     } while (win == -1 || win == 0);
00185     return 0;
00186 }

```

## 7.3 connect4.cpp File Reference

Implementation of [connect4.h](#).

```
#include "connect4.h"
```

### Functions

- ostream & operator<< (ostream &os, const [Connect4](#) &c)

### 7.3.1 Detailed Description

Implementation of [connect4.h](#).

Author

Mirko Laruina

Date

2020-05-14

See also

[connect4.h](#)

Definition in file [connect4.cpp](#).

## 7.4 connect4.cpp

```

00001
00012 #include "connect4.h"
00013 using namespace std;
00014
00015 void Connect4::print(ostream& os){
00016     os<<"this";
00017 }
00018
00019 Connect4::Connect4(int rows /* = 6 */, int columns /* = 7 */){
00020     rows_ = rows;
00021     cols_ = columns;
00022     size_ = rows*columns;
00023     full_ = false;
00024
00025     //Maybe check for overflow if we will use different board values
00026
00027     cells_ = new char[size_];
00028     memset(cells_, 0, size_);
00029 }
00030
00031 int8_t Connect4::play(int col, char player){
00032     // bool col_full = true;
00033     if(player == 0){
00034         player = player_;
00035     }
00036
00037     //Trying to play with a full board
00038     if(full_){
00039         return -2;
00040     }
00041
00042     for(int i = rows_-1; i>=0; --i){
00043         if(cells_[i*cols_+col] == 0){
00044             // col_full = false;
00045             cells_[i*cols_+col] = player;
00046             if( checkWin(i, col, player) ){
00047                 return 1;
00048             } else {
00049                 //All the board could be full now
00050                 if(i == 0 && checkFullTopRow()){
00051                     full_ = true;
00052                     return -2;
00053                 } else {
00054                     return 0;
00055                 }
00056             }
00057         }
00058     }
00059
00060     //We are sure the board is not full, otherwise we would have already exited
00061     //If a play was possible, we would have already exited too
00062     //Only possible case is full column

```

```

00063     return -1;
00064 }
00065
00066 int Connect4::countNexts(char player, int row, int col, int di, int dj){
00067     int count = 0;
00068     for(
00069         int i = row+di, j = col+dj;
00070         i >= 0 && j >= 0 && i < rows_ && j < cols_;
00071         i+=di, j+=dj)
00072     {
00073         if(cells_[i*cols_+j] != player){
00074             break;
00075         } else {
00076             LOG(LOG_DEBUG, "%d %d", i, j);
00077             count++;
00078         }
00079     }
00080     return count;
00081 }
00082
00083 bool Connect4::checkWin(int row, int col, char player){
00084     /*
00085         Take any of the 4 possible directions
00086         count how many token of the same player there are
00087         before and after the new one
00088         if more than 4, declare win
00089     */
00090
00091     LOG(LOG_DEBUG, "Checking (%d, %d)", row, col);
00092     if(player == 0){
00093         player = player_;
00094     }
00095
00096     for(int di = 1; di >= 0 && di != -1; --di){
00097         for(int dj = 1; dj >= 0 && di != -1; --dj){
00098             // direction (0, 0) is useless, since we would miss diagonal (-1, 1)
00099             // we can exploit the loop to iterate over that
00100             if(di == 0 && dj == 0){
00101                 di = -1;
00102                 dj = 1;
00103             }
00104
00105             int count_forward = Connect4::countNexts(player, row, col, di, dj);
00106             int count_backward = Connect4::countNexts(player, row, col, -di, -dj);
00107
00108             // N_IN_A_ROW minus 1 since the token just inserted is excluded
00109             if(count_forward + count_backward >= (N_IN_A_ROW - 1)){
00110                 return true;
00111             }
00112         }
00113     }
00114
00115     return false;
00116 }
00117 }
00118
00119 bool Connect4::checkFullTopRow(){
00120     for(int j = 0; j<cols_; ++j){
00121         if(cells_[j] == 0){
00122             return false;
00123         }
00124     }
00125     return true;
00126 }
00127
00128 int Connect4::getNumCols(){
00129     return cols_;
00130 }
00131
00132 bool Connect4::setPlayer(char player){
00133     if(player == 'X' || player == 'x'
00134        || player == 'O' || player == 'o'){
00135         player_ = toupper(player);
00136         adversary_ = player_ == 'X' ? 'O' : 'X';
00137         return true;
00138     }
00139     return false;
00140 }
00141
00142 char Connect4::getPlayer(){
00143     return player_;
00144 }
00145
00146 char Connect4::getAdv(){
00147     return adversary_;
00148 }
00149

```

```

00150 ostream& operator<<(ostream& os, const Connect4& c){
00151     int width = 2+3*(c.rows_+1);
00152     for(int i = 0; i<width; ++i){
00153         os<<'*';
00154     }
00155     os<<endl;
00156
00157     for(int i = 0; i<c.rows_; ++i){
00158         os<<"*";
00159         for(int j = 0; j<c.cols_; ++j){
00160             if(c.cells_[i*c.cols_+j] == 0){
00161                 os<<" ";
00162             } else {
00163                 os<<" " << (c.cells_[i*c.cols_+j] == 'X' ? "\033[31mX" : "\033[34mO") <<" ";
00164             }
00165         }
00166         os<<"\033[0m*"<<endl;
00167     }
00168
00169     for(int i = 0; i<width; ++i){
00170         os<<'*';
00171     }
00172     os<<endl;
00173
00174     for(int i = 0; i<width; ++i){
00175         if( (i+1)%3 == 0 ){
00176             os<<(i+1)/3;
00177         } else {
00178             os<<" ";
00179         }
00180     }
00181     os<<endl;
00182     return os;
00183 }

```

## 7.5 connect4.h File Reference

Header file for the class responsible of handling the board of a [Connect4](#) game.

```

#include <iostream>
#include <cstring>
#include "logging.h"

```

### Data Structures

- class [Connect4](#)

### Macros

- #define `N_IN_A_ROW` 4

#### 7.5.1 Detailed Description

Header file for the class responsible of handling the board of a [Connect4](#) game.

#### Author

Mirko Laruina

#### Date

2020-05-14

Definition in file [connect4.h](#).



## 7.6 connect4.h

```

00001
00010 #ifndef CONNECT4_H
00011 #define CONNECT4_H
00012 #include <iostream>
00013 #include <cstring>
00014 #include "logging.h"
00015
00016 #define N_IN_A_ROW 4
00017
00018 class Connect4 {
00020     int rows_, cols_, size_;
00021
00023     bool full_;
00024
00026     char* cells_;
00027
00029     char player_;
00030
00032     char adversary_;
00033
00046     int countNexts(char player, int row, int col, int di, int dj);
00047
00054     bool checkFullTopRow();
00055     public:
00056
00063     Connect4(int rows = 6, int columns = 7);
00064
00070     int getNumCols();
00071
00083     int8_t play(int column, char player = 0);
00084
00094     bool checkWin(int starting_row, int starting_col, char player = 0);
00095
00103     bool setPlayer(char player);
00104
00110     char getPlayer();
00111
00117     char getAdv();
00118
00124     void print(std::ostream& os);
00125
00126     friend std::ostream& operator<<(std::ostream& os, const Connect4& b);
00127 };
00128 #endif //CONNECT4_H

```

## 7.7 dump\_buffer.cpp File Reference

Implementation of [dump\\_buffer.h](#).

```

#include "utils/dump_buffer.h"
#include "logging.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

### Functions

- void [dump\\_buffer\\_hex](#) (char \*buffer, int len)  
*Prints content of buffer to stdout, showing it as hex values.*

#### 7.7.1 Detailed Description

Implementation of [dump\\_buffer.h](#).

**Author**

Riccardo Mancini

**See also**[dump\\_buffer.h](#)Definition in file [dump\\_buffer.cpp](#).**7.7.2 Function Documentation**

**7.7.2.1 dump\_buffer\_hex()** void dump\_buffer\_hex (

```

    char * buffer,
    int len )

```

Prints content of buffer to stdout, showing it as hex values.

It uses the logging infrastructure to print.

**Parameters**

<i>buffer</i>	pointer to the buffer to be printed
<i>len</i>	the length (in bytes) of the buffer

Definition at line 17 of file [dump\\_buffer.cpp](#).**7.8 dump\_buffer.cpp**

```

00001
00010 #include "utils/dump_buffer.h"
00011 #include "logging.h"
00012 #include <stdio.h>
00013 #include <stdlib.h>
00014 #include <string.h>
00015
00016
00017 void dump_buffer_hex(char* buffer, int len){
00018     char *str, tmp[4];
00019     int i;
00020
00021     str = (char*) malloc(len*3+1);
00022
00023     str[0] = '\0';
00024     for (i=0; i<len; i++){
00025         sprintf(tmp, "%02x ", (unsigned char) buffer[i]);
00026         strcat(str, tmp);
00027     }
00028
00029     LOG(LOG_DEBUG, "%s", str);
00030     free(str);
00031 }

```

**7.9 dump\_buffer.h File Reference**

Utility function for dumping a buffer as hex string.

## Functions

- void [dump\\_buffer\\_hex](#) (char \*buffer, int len)  
*Prints content of buffer to stdout, showing it as hex values.*

### 7.9.1 Detailed Description

Utility function for dumping a buffer as hex string.

#### Author

Riccardo Mancini

#### Date

2020-05-17

Definition in file [dump\\_buffer.h](#).

### 7.9.2 Function Documentation

**7.9.2.1 dump\_buffer\_hex()** void dump\_buffer\_hex (  
    char \* *buffer*,  
    int *len* )

Prints content of buffer to stdout, showing it as hex values.

It uses the logging infrastructure to print.

#### Parameters

<i>buffer</i>	pointer to the buffer to be printed
<i>len</i>	the length (in bytes) of the buffer

Definition at line 17 of file [dump\\_buffer.cpp](#).

## 7.10 dump\_buffer.h

```
00001
00010 #ifndef DUMP_BUFFER_H
00011 #define DUMP_BUFFER_H
00012
00013
00022 void dump_buffer_hex(char* buffer, int len);
00023
00024
00025 #endif // DUMP_BUFFER_H
```

## 7.11 host.h File Reference

Definition of the helper class "Host".

```
#include "logging.h"
#include "network/inet_utils.h"
#include "network/messages.h"
```

### Data Structures

- class [Host](#)

*Class that holds a host information.*

#### 7.11.1 Detailed Description

Definition of the helper class "Host".

##### Author

Riccardo Mancini

##### Date

2020-05-17

Definition in file [host.h](#).

## 7.12 host.h

```
00001
00010 #ifndef HOST_H
00011 #define HOST_H
00012
00013 #include "logging.h"
00014 #include "network/inet_utils.h"
00015 #include "network/messages.h"
00016
00023 class Host{
00024 private:
00025     struct sockaddr_in addr;
00026
00027 public:
00033     Host(struct sockaddr_in addr)
00034         : addr(addr) {}
00035
00041     Host(char* ip, int port){
00042         addr = make_sv_sockaddr_in(ip, port);
00043     }
00044
00046     struct sockaddr_in getAddress(){return addr;}
00047
00049     string toString(){
00050         return sockaddr_in_to_string(addr);
00051     }
00052
00053 };
00054
00055 #endif // HOST_H
```

## 7.13 inet\_utils.cpp File Reference

Implementation of [inet\\_utils.h](#).

```
#include <stdlib.h>
#include <string>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "network/inet_utils.h"
#include "logging.h"
```

### Functions

- int [bind\\_random\\_port](#) (int socket, struct sockaddr\_in \*addr)  
*Binds socket to a random port.*
- struct sockaddr\_in [make\\_sv\\_sockaddr\\_in](#) (char \*ip, int port)  
*Makes sockaddr\_in structure given ip string and port of server.*
- struct sockaddr\_in [make\\_my\\_sockaddr\\_in](#) (int port)  
*Makes sockaddr\_in structure of this host.*
- int [sockaddr\\_in\\_cmp](#) (struct sockaddr\_in sai1, struct sockaddr\_in sai2)  
*Compares INET addresses, returning 0 in case they're equal.*
- string [sockaddr\\_in\\_to\\_string](#) (struct sockaddr\_in src)  
*Converts sockaddr\_in structure to string to be printed.*

### 7.13.1 Detailed Description

Implementation of [inet\\_utils.h](#).

#### Author

Riccardo Mancini

#### See also

[inet\\_utils.h](#)

#### Date

2020-05-17

Definition in file [inet\\_utils.cpp](#).

### 7.13.2 Function Documentation

**7.13.2.1 bind\_random\_port()** int bind\_random\_port (  
    int socket,  
    struct sockaddr\_in \* addr )

Binds socket to a random port.

## Parameters

<i>socket</i>	socket ID
<i>addr</i>	inet addr structure

## Returns

0 in case of failure, port it could bind to otherwise

## See also

[FROM\\_PORT](#)

[TO\\_PORT](#)

[MAX\\_TRIES](#)

Definition at line 24 of file [inet\\_utils.cpp](#).

**7.13.2.2 make\_my\_sockaddr\_in()** `struct sockaddr_in make_my_sockaddr_in (`  
`int port )`

Makes sockaddr\_in structure of this host.

INADDR\_ANY is used as IP address.

## Parameters

<i>port</i>	port of the server
-------------	--------------------

## Returns

sockaddr\_in structure this host on given port

Definition at line 53 of file [inet\\_utils.cpp](#).

**7.13.2.3 make\_sv\_sockaddr\_in()** `struct sockaddr_in make_sv_sockaddr_in (`  
`char * ip,`  
`int port )`

Makes sockaddr\_in structure given ip string and port of server.

## Parameters

<i>ip</i>	ip address of server
<i>port</i>	port of the server

### Returns

sockaddr\_in structure for the given server

Definition at line 44 of file [inet\\_utils.cpp](#).

```
7.13.2.4 sockaddr_in_cmp() int sockaddr_in_cmp (
    struct sockaddr_in sai1,
    struct sockaddr_in sai2 )
```

Compares INET addresses, returning 0 in case they're equal.

### Parameters

<i>sai1</i>	first address
<i>sai2</i>	second address

### Returns

0 if they're equal, 1 otherwise

Definition at line 62 of file [inet\\_utils.cpp](#).

```
7.13.2.5 sockaddr_in_to_string() string sockaddr_in_to_string (
    struct sockaddr_in src )
```

Converts sockaddr\_in structure to string to be printed.

### Parameters

<i>src</i>	the input address
<i>dst</i>	the output string (must be at least MAX_SOCKADDR_STR_LEN long)

Definition at line 70 of file [inet\\_utils.cpp](#).

## 7.14 inet\_utils.cpp

```
00001
00012 #include <stdlib.h>
00013 #include <string>
00014 #include <string.h>
00015 #include <sys/socket.h>
00016 #include <netinet/in.h>
00017 #include <arpa/inet.h>
00018
00019 #include "network/inet_utils.h"
00020 #include "logging.h"
00021
00022 using namespace std;
00023
00024 int bind_random_port(int socket, struct sockaddr_in *addr){
```

```

00025     int port, ret, i;
00026     for (i = 0; i < MAX_TRIES; i++){
00027         if (i == 0) // first I generate a random one
00028             port = rand() % (TO_PORT - FROM_PORT + 1) + FROM_PORT;
00029         else //if it's not free I scan the next one
00030             port = (port - FROM_PORT + 1) % (TO_PORT - FROM_PORT + 1) + FROM_PORT;
00031
00032         LOG(LOG_DEBUG, "Trying port %d...", port);
00033
00034         addr->sin_port = htons(port);
00035         ret = bind(socket, (struct sockaddr *)addr, sizeof(*addr));
00036         if (ret != -1)
00037             return port;
00038         // consider only some errors?
00039     }
00040     LOG(LOG_ERR, "Could not bind to random port after %d attempts", MAX_TRIES);
00041     return 0;
00042 }
00043
00044 struct sockaddr_in make_sv_sockaddr_in(char *ip, int port){
00045     struct sockaddr_in addr;
00046     memset(&addr, 0, sizeof(addr));
00047     addr.sin_family = AF_INET;
00048     addr.sin_port = htons(port);
00049     inet_pton(AF_INET, ip, &addr.sin_addr);
00050     return addr;
00051 }
00052
00053 struct sockaddr_in make_my_sockaddr_in(int port){
00054     struct sockaddr_in addr;
00055     memset(&addr, 0, sizeof(addr));
00056     addr.sin_family = AF_INET;
00057     addr.sin_port = htons(port);
00058     addr.sin_addr.s_addr = htonl(INADDR_ANY);
00059     return addr;
00060 }
00061
00062 int sockaddr_in_cmp(struct sockaddr_in sai1, struct sockaddr_in sai2){
00063     if (sai1.sin_port == sai2.sin_port &&
00064         sai1.sin_addr.s_addr == sai2.sin_addr.s_addr)
00065         return 0;
00066     else
00067         return 1;
00068 }
00069
00070 string sockaddr_in_to_string(struct sockaddr_in src){
00071     char dst[MAX_SOCKADDR_STR_LEN];
00072     char port_str[6];
00073     const char *ret;
00074
00075     sprintf(port_str, "%d", ntohs(src.sin_port));
00076
00077     ret = inet_ntop(AF_INET, (void *)&src.sin_addr, dst, MAX_SOCKADDR_STR_LEN);
00078     if (ret != NULL){
00079         strcat(dst, ":");
00080         strcat(dst, port_str);
00081     } else {
00082         strcpy(dst, "ERROR");
00083     }
00084
00085     string s = dst;
00086
00087     return s;
00088 }

```

## 7.15 inet\_utils.h File Reference

Utility functions for managing inet addresses.

```

#include <sys/socket.h>
#include <netinet/in.h>
#include <string>

```

### Macros

- #define FROM\_PORT 49152



- Random port will be greater or equal to FROM\_PORT.*
  - `#define TO_PORT 65535`  
*Random port will be lower or equal to TO\_PORT.*
- `#define MAX_TRIES 256`  
*Maximum number of trials before giving up opening a random port.*
- `#define MAX_SOCKADDR_STR_LEN 22`  
*Maximum number of characters of INET address to string (eg 123.156.189.123:45678).*

## Functions

- `int bind_random_port (int socket, struct sockaddr_in *addr)`  
*Binds socket to a random port.*
- `struct sockaddr_in make_sv_sockaddr_in (char *ip, int port)`  
*Makes sockaddr\_in structure given ip string and port of server.*
- `struct sockaddr_in make_my_sockaddr_in (int port)`  
*Makes sockaddr\_in structure of this host.*
- `int sockaddr_in_cmp (struct sockaddr_in sai1, struct sockaddr_in sai2)`  
*Compares INET addresses, returning 0 in case they're equal.*
- `string sockaddr_in_to_string (struct sockaddr_in src)`  
*Converts sockaddr\_in structure to string to be printed.*

### 7.15.1 Detailed Description

Utility functions for managing inet addresses.

#### Author

Riccardo Mancini

This library provides functions for creating `sockaddr_in` structures from IP address string and integer port number and for binding to a random port (chosen using `rand()` builtin C function).

#### Date

2020-05-17

#### See also

`sockaddr_in`  
`rand`

Definition in file [inet\\_utils.h](#).

### 7.15.2 Function Documentation

**7.15.2.1 `bind_random_port()`** `int bind_random_port (`  
    `int socket,`  
    `struct sockaddr_in * addr )`

Binds socket to a random port.

## Parameters

<i>socket</i>	socket ID
<i>addr</i>	inet addr structure

## Returns

0 in case of failure, port it could bind to otherwise

## See also

[FROM\\_PORT](#)

[TO\\_PORT](#)

[MAX\\_TRIES](#)

Definition at line 24 of file [inet\\_utils.cpp](#).

**7.15.2.2 make\_my\_sockaddr\_in()** `struct sockaddr_in make_my_sockaddr_in (`  
`int port )`

Makes sockaddr\_in structure of this host.

INADDR\_ANY is used as IP address.

## Parameters

<i>port</i>	port of the server
-------------	--------------------

## Returns

sockaddr\_in structure this host on given port

Definition at line 53 of file [inet\\_utils.cpp](#).

**7.15.2.3 make\_sv\_sockaddr\_in()** `struct sockaddr_in make_sv_sockaddr_in (`  
`char * ip,`  
`int port )`

Makes sockaddr\_in structure given ip string and port of server.

## Parameters

<i>ip</i>	ip address of server
<i>port</i>	port of the server

**Returns**

sockaddr\_in structure for the given server

Definition at line 44 of file [inet\\_utils.cpp](#).

```
7.15.2.4 sockaddr_in_cmp()  int sockaddr_in_cmp (
    struct sockaddr_in sai1,
    struct sockaddr_in sai2 )
```

Compares INET addresses, returning 0 in case they're equal.

**Parameters**

<i>sai1</i>	first address
<i>sai2</i>	second address

**Returns**

0 if they're equal, 1 otherwise

Definition at line 62 of file [inet\\_utils.cpp](#).

```
7.15.2.5 sockaddr_in_to_string()  string sockaddr_in_to_string (
    struct sockaddr_in src )
```

Converts sockaddr\_in structure to string to be printed.

**Parameters**

<i>src</i>	the input address
<i>dst</i>	the output string (must be at least MAX_SOCKADDR_STR_LEN long)

Definition at line 70 of file [inet\\_utils.cpp](#).

## 7.16 inet\_utils.h

```
00001
00017 #ifndef INET_UTILS
00018 #define INET_UTILS
00019
00020
00021 #include <sys/socket.h>
00022 #include <netinet/in.h>
00023 #include <string>
00024
00025 using namespace std;
00026
00028 #define FROM_PORT 49152
00029
00031 #define TO_PORT 65535
```

```
00032
00034 #define MAX_TRIES 256
00035
00040 #define MAX_SOCKADDR_STR_LEN 22
00041
00042
00054 int bind_random_port(int socket, struct sockaddr_in *addr);
00055
00063 struct sockaddr_in make_sv_sockaddr_in(char* ip, int port);
00064
00073 struct sockaddr_in make_my_sockaddr_in(int port);
00074
00082 int sockaddr_in_cmp(struct sockaddr_in sail, struct sockaddr_in sai2);
00083
00090 string sockaddr_in_to_string(struct sockaddr_in src);
00091
00092
00093 #endif
```

## 7.17 logging.h File Reference

Logging macro.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
```

### Macros

- #define **LOG\_FATAL** (1)
- #define **LOG\_ERR** (2)
- #define **LOG\_WARN** (3)
- #define **LOG\_INFO** (4)
- #define **LOG\_DEBUG** (5)
- #define **LOG\_LEVEL** LOG\_DEBUG
- #define **LOG**(level, ...)

### 7.17.1 Detailed Description

Logging macro.

#### Author

Riccardo Mancini

This file contains a macro for logging in different levels.

There are 5 levels of logging:

- fatal (LOG\_FATAL)
- error (LOG\_ERROR)
- warning (LOG\_WARN)
- information (LOG\_INFO)
- debug (LOG\_DEBUG)

The first three will be outputted to stderr, the latter two to stdout.

You can define a LOG\_LEVEL for hiding some of the logging messages in a per-executable basis. In order to do so, you need to put

```
const int LOG_LEVEL = LOG_INFO;
```

in the file containing the main and

```
extern const int LOG_LEVEL;
```

in any other file using this macro.

Adapted from <https://stackoverflow.com/a/328660>

Definition in file [logging.h](#).

## 7.18 logging.h

```
00001
00033 #ifndef LOGGING
00034 #define LOGGING
00035
00036
00037 #include <stdio.h>
00038 #include <sys/types.h>
00039 #include <unistd.h>
00040
00041
00042 #define LOG_FATAL      (1)
00043 #define LOG_ERR        (2)
00044 #define LOG_WARN       (3)
00045 #define LOG_INFO       (4)
00046 #define LOG_DEBUG      (5)
00047
00048 #ifndef LOG_LEVEL
00049 #define LOG_LEVEL LOG_DEBUG
00050 #endif
00051
00052
00053 #define LOG(level, ...) do { \
00054     if (level <= LOG_LEVEL) { \
00055         FILE *dbgstream; \
00056         char where[35]; \
00057         switch(level){ \
00058             case LOG_FATAL: \
00059                 dbgstream = stderr; \
00060                 fprintf(dbgstream, "[FATAL]"); \
00061                 break; \
00062             case LOG_ERR: \
00063                 dbgstream = stderr; \
00064                 fprintf(dbgstream, "[ERROR]"); \
00065                 break; \
00066             case LOG_WARN: \
00067                 dbgstream = stderr; \
00068                 fprintf(dbgstream, "[WARN ]"); \
00069                 break; \
00070             case LOG_INFO: \
00071                 dbgstream = stdout; \
00072                 fprintf(dbgstream, "[INFO ]"); \
00073                 break; \
00074             case LOG_DEBUG: \
00075                 dbgstream = stdout; \
00076                 fprintf(dbgstream, "[DEBUG]"); \
00077                 break; \
00078             } \
00079             fprintf(dbgstream, "[%5d]", (int) getpid()); \
00080             snprintf(where, 35, "%s:%d", __FILE__, __LINE__); \
00081             fprintf(dbgstream, " %-25s ", where); \
00082             fprintf(dbgstream, __VA_ARGS__); \
00083             fprintf(dbgstream, "\n"); \
00084             fflush(dbgstream); \
00085         } \
00086     } while (0)
00087
00088
00089 #endif
```

## 7.19 messages.cpp File Reference

Implementation of [messages.h](#).

```
#include <cstdlib>
#include "network/messages.h"
#include "utils/dump_buffer.h"
```

### Functions

- [Message](#) \* [readMessage](#) (char \*buffer, int len)  
*Reads the message using the correct class and returns a pointer to it.*

#### 7.19.1 Detailed Description

Implementation of [messages.h](#).

#### Author

Riccardo Mancini

#### See also

[messages.h](#)

Definition in file [messages.cpp](#).

#### 7.19.2 Function Documentation

**7.19.2.1 readMessage()** [Message](#)\* readMessage (  
    char \* *buffer*,  
    int *len* )

Reads the message using the correct class and returns a pointer to it.

NB: remeber to dispose of the created [Message](#) when you are done with it.

Definition at line 15 of file [messages.cpp](#).

## 7.20 messages.cpp

```

00001
00010 #include <cstdlib>
00011
00012 #include "network/messages.h"
00013 #include "utils/dump_buffer.h"
00014
00015 Message* readMessage(char *buffer, int len){
00016     Message *m;
00017     int ret;
00018
00019     switch(buffer[0]){
00020         case START_GAME:
00021             m = new StartGameMessage;
00022             break;
00023         case MOVE:
00024             m = new MoveMessage;
00025             break;
00026         default:
00027             m = NULL;
00028             LOG(LOG_ERR, "Unrecognized message type %d", buffer[0]);
00029             dump_buffer_hex(buffer, len);
00030             return NULL;
00031     };
00032
00033     ret = m->read(buffer, len);
00034
00035     if (ret != 0){
00036         LOG(LOG_ERR, "Error reading message of type %d: %d", buffer[0], ret);
00037         dump_buffer_hex(buffer, len);
00038         return NULL;
00039     } else{
00040         return m;
00041     }
00042 }
00043
00044
00045 int StartGameMessage::write(char *buffer){
00046     buffer[0] = (char) START_GAME;
00047     return 1;
00048 }
00049
00050 int StartGameMessage::read(char *buffer, int len){
00051     return 0;
00052 }
00053
00054 int MoveMessage::write(char *buffer){
00055     buffer[0] = (char) MOVE;
00056     buffer[1] = col;
00057     return 2;
00058 }
00059
00060 int MoveMessage::read(char *buffer, int len){
00061     if (len < 2)
00062         return 1;
00063
00064     col = buffer[1];
00065     return 0;
00066 }

```

## 7.21 messages.h File Reference

Definition of messages.

```

#include "logging.h"
#include <string>
#include "network/inet_utils.h"

```

### Data Structures

- class [Message](#)  
*Abstract class for Messages.*
- class [StartGameMessage](#)

- Message that signals to start a new game.*
- class [MoveMessage](#)  
*Message that signals a move.*

## Macros

- #define [MAX\\_MSG\\_SIZE](#) 1024  
*Maximum message size.*

## Enumerations

- enum [MessageType](#) { [START\\_GAME](#), [MOVE](#) }  
*Possible type of messages.*

## Functions

- [Message \\*](#) [readMessage](#) (char \*buffer, int len)  
*Reads the message using the correct class and returns a pointer to it.*

### 7.21.1 Detailed Description

Definition of messages.

#### Author

Riccardo Mancini

#### Date

2020-05-17

Definition in file [messages.h](#).

### 7.21.2 Macro Definition Documentation

#### 7.21.2.1 [MAX\\_MSG\\_SIZE](#) #define [MAX\\_MSG\\_SIZE](#) 1024

Maximum message size.

TODO: it is random, calculate it

Definition at line 24 of file [messages.h](#).



### 7.21.3 Enumeration Type Documentation

#### 7.21.3.1 `MessageType` `enum MessageType`

Possible type of messages.

When adding a new message class, add a related type here and set its `getType` method to return it.

Definition at line 32 of file `messages.h`.

### 7.21.4 Function Documentation

#### 7.21.4.1 `readMessage()` `Message* readMessage (` `char * buffer,` `int len )`

Reads the message using the correct class and returns a pointer to it.

NB: remeber to dispose of the created `Message` when you are done with it.

Definition at line 15 of file `messages.cpp`.

## 7.22 `messages.h`

```
00001
00010 #ifndef MESSAGES_H
00011 #define MESSAGES_H
00012
00013 #include "logging.h"
00014 #include <string>
00015 #include "network/inet_utils.h"
00016
00017 using namespace std;
00018
00024 #define MAX_MSG_SIZE 1024
00025
00032 enum MessageType {START_GAME, MOVE};
00033
00037 class Message{
00038 public:
00044     virtual int write(char *buffer) = 0;
00045
00052     virtual int read(char *buffer, int len) = 0;
00053
00057     virtual size_t size() = 0;
00058
00062     virtual string getName() = 0;
00063
00064     virtual MessageType getType() = 0;
00065 };
00066
00070 class StartGameMessage : public Message{
00071 public:
00072     StartGameMessage() {}
00073
00074     int write(char *buffer);
00075     int read(char *buffer, int len);
00076
00077     size_t size(){return 1;}
00078
00079     string getName(){return "StartGame";}
```

```

00080
00081     MessageType getType() {return START_GAME;}
00082 };
00083
00087 class MoveMessage : public Message{
00088 private:
00089     char col;
00090 public:
00091     MoveMessage() {}
00092     MoveMessage(char col) : col(col) {}
00093
00094     int write(char *buffer);
00095     int read(char *buffer, int len);
00096
00097     size_t size() {return 2;}
00098
00099     string getName() {return "Move";}
00100
00101     char getColumn() {return col;}
00102
00103     MessageType getType() {return MOVE;}
00104 };
00105
00111 Message* readMessage(char *buffer, int len);
00112
00113 #endif // MESSAGES_H

```

## 7.23 single\_player.cpp File Reference

Implementation of a 4-in-a-row game.

```

#include <iostream>
#include <cstdlib>
#include <ctime>
#include "connect4.h"
#include "logging.h"

```

### Functions

- void **printWelcome** ()
- int **main** ()

### Variables

- char **players** [] = {'X', 'O'}

#### 7.23.1 Detailed Description

Implementation of a 4-in-a-row game.

#### Author

Mirko Laruina

#### Date

2020-05-17

Definition in file [single\\_player.cpp](#).

## 7.24 single\_player.cpp

```

00001
00009 #include <iostream>
00010 #include <cstdlib>
00011 #include <ctime>
00012 #include "connect4.h"
00013 #include "logging.h"
00014
00015 using namespace std;
00016
00017 char players[] = {'X', 'O'};
00018
00019 void printWelcome(){
00020     cout<<"*****\n"
00021         <<"*
00022         <<"*
00023         <<"*
00024         <<"*
00025         <<"*
00026         <<"*
00027         <<"*****\n"
00028         <<endl;
00029 }
00030
00031 int main(){
00032     // handle server selection on params here
00033     char in_buffer[256];
00034     int choosen_col;
00035     int adv_col;
00036     int win;
00037     srand(time(NULL));
00038
00039     printWelcome();
00040     cout<<endl<<"Welcome to 4-in-a-row!"<<endl;
00041     cout<<"The rules of the game are simple: you win when you have 4 connected tokens along any
direction."<<endl;
00042
00043     Connect4 c;
00044     cout<<"Who do you want to be? X or O ?"<<endl;
00045
00046     do {
00047         cout<<"> ";
00048         cin.getline(in_buffer, sizeof(in_buffer));
00049     } while (!c.setPlayer(in_buffer[0]));
00050     cout<<"You are playing as "<<c.getPlayer()<<endl;
00051     cout<<"This is the starting board:"<<endl;
00052     cout<<c;
00053
00054     do {
00055         cout<<"Write the column you want to insert the token to"<<endl;
00056         do {
00057             cout<<"> ";
00058             cin.getline(in_buffer, sizeof(in_buffer));
00059             choosen_col = in_buffer[0]-'0';
00060         } while(choosen_col < 0 || choosen_col > 7);
00061
00062         win = c.play(choosen_col-1, c.getPlayer());
00063         cout<<c;
00064         if(win == 1){
00065             cout<<"Congratulation, you won!"<<endl;
00066         } else if(win == -1){
00067             cout<<"The column is full, choose a different one!"<<endl;
00068             continue;
00069         } else if(win == -2){
00070             cout<<"The entire board is filled: it is a draw!"<<endl;
00071             break;
00072         }
00073
00074         if(win != 1){
00075             do {
00076                 adv_col = rand()%c.getNumCols();
00077                 cout<<"Your enemy has chosen column "<<adv_col<<endl;
00078                 win = c.play(adv_col, c.getAdv());
00079                 cout<<c;
00080                 if(win == 1){
00081                     cout<<"Damn! You lost!"<<endl;
00082                 } else if(win == -1){
00083                     cout<<"The column is full, the adversary has to chose a different one!"<<endl;
00084                     continue;
00085                 } else if(win == -2){
00086                     cout<<"The entire board is filled: it is a draw!"<<endl;
00087                     break;
00088                 }
00089             } while (win == -1);
00090         }
00091     } while (win == -1 || win == 0);

```

```
00092     return 0;
00093 }
```

## 7.25 socket\_wrapper.cpp File Reference

Implementation of [socket\\_wrapper.h](#).

```
#include "logging.h"
#include "network/socket_wrapper.h"
```

### 7.25.1 Detailed Description

Implementation of [socket\\_wrapper.h](#).

#### Author

Riccardo Mancini

#### See also

[socket\\_wrapper.h](#)

Definition in file [socket\\_wrapper.cpp](#).

## 7.26 socket\_wrapper.cpp

```
00001
00010 #include "logging.h"
00011 #include "network/socket_wrapper.h"
00012
00013 SocketWrapper::SocketWrapper() {
00014     socket_fd = socket(AF_INET, SOCK_STREAM, 0);
00015     if (socket_fd < 0){
00016         LOG(LOG_ERR, "Could not create socket!\n");
00017         perror("Error: ");
00018         return;
00019     }
00020 }
00021
00022 Message* SocketWrapper::receiveAnyMsg(size_t size){
00023     int len;
00024     char* in_buffer;
00025
00026     in_buffer = (char*) malloc(size);
00027
00028     len = recv(socket_fd, in_buffer, size, 0);
00029
00030     Message *m = readMessage(in_buffer, len);
00031
00032     free(in_buffer);
00033
00034     return m;
00035 }
00036
00037 Message* SocketWrapper::receiveMsg(MessageType type, size_t size /*=MAX_MSG_SIZE*/){
00038     return this->receiveMsg(&type, 1, size);
00039 }
00040
00041 Message* SocketWrapper::receiveMsg(MessageType type[], int n_types,
00042                                     size_t size /*=MAX_MSG_SIZE*/){
00043     Message *m = NULL;
00044     while (m == NULL){
00045         m = this->receiveAnyMsg();
00046         if (m != NULL){
00047             for (int i = 0; i < n_types; i++){
```

```

00048         if (m->getType() == type[i]){
00049             return m;
00050         }
00051         LOG(LOG_WARN, "Received unexpected message of type %d", m->getType());
00052     }
00053 }
00054 }
00055 //TODO: add timeout?
00056 return NULL;
00057 }
00058
00059
00060 int SocketWrapper::sendMsg(Message *msg){
00061     int msglen, len;
00062     char *out_buffer;
00063
00064     msglen = msg->size();
00065     out_buffer = (char*) malloc(msglen);
00066
00067     msg->write(out_buffer);
00068
00069     len = send(socket_fd, out_buffer, msglen, 0);
00070     if (len != msglen){
00071         LOG(LOG_ERR, "Error sending %s: len (%d) != msglen (%d)",
00072             msg->getName().c_str(),
00073             len,
00074             msglen
00075         );
00076         return 1;
00077     }
00078
00079     LOG(LOG_DEBUG, "Sent message %s", msg->getName().c_str());
00080
00081     free(out_buffer);
00082     return 0;
00083 }
00084
00085 int ClientSocketWrapper::connectServer(Host host){
00086     int ret;
00087
00088     other_addr = host.getAddress();
00089
00090     ret = connect(
00091         socket_fd,
00092         (struct sockaddr*) &other_addr,
00093         sizeof(other_addr)
00094     );
00095
00096     if (ret != 0){
00097         LOG(LOG_ERR, "Error connecting to %s",
00098             sockaddr_in_to_string(host.getAddress()).c_str());
00099         perror("Error: ");
00100         return ret;
00101     }
00102
00103     return ret;
00104 }
00105
00106 ServerSocketWrapper::ServerSocketWrapper(){
00107     my_addr = make_my_sockaddr_in(0);
00108     int ret = bind_random_port(socket_fd, &my_addr);
00109     if (ret <= 0){
00110         LOG(LOG_ERR, "Error in binding\n");
00111         perror("Error: ");
00112     }
00113
00114     ret = listen(socket_fd, 10);
00115     if (ret != 0){
00116         LOG(LOG_ERR, "Error in setting socket to listen mode\n");
00117         perror("Error: ");
00118     }
00119 }
00120
00121 ServerSocketWrapper::ServerSocketWrapper(int port){
00122     my_addr = make_my_sockaddr_in(port);
00123     int ret = bind(socket_fd, (struct sockaddr*) &my_addr, sizeof(my_addr));
00124     if (ret != 0){
00125         LOG(LOG_ERR, "Error in binding\n");
00126         perror("Error: ");
00127     }
00128
00129     ret = listen(socket_fd, 10);
00130     if (ret != 0){
00131         LOG(LOG_ERR, "Error in setting socket to listen mode\n");
00132         perror("Error: ");
00133     }
00134 }

```

```
00135
00136 SocketWrapper* ServerSocketWrapper::acceptClient() {
00137     socklen_t len = sizeof(other_addr);
00138     int new_sd = accept(
00139         socket_fd,
00140         (struct sockaddr*) &other_addr,
00141         &len
00142     );
00143
00144     SocketWrapper *sw = new SocketWrapper(new_sd);
00145     sw->setOtherAddr(other_addr);
00146     return sw;
00147 }
```

## 7.27 socket\_wrapper.h File Reference

Definition of the helper class "SocketWrapper" and derivatives.

```
#include "logging.h"
#include "network/inet_utils.h"
#include "network/messages.h"
#include "network/host.h"
```

### Data Structures

- class [SocketWrapper](#)  
*Wrapper class around sockaddr\_in and socket descriptor.*
- class [ClientSocketWrapper](#)  
*SocketWrapper for a TCP client.*
- class [ServerSocketWrapper](#)  
*SocketWrapper for a TCP server.*

### 7.27.1 Detailed Description

Definition of the helper class "SocketWrapper" and derivatives.

#### Author

Riccardo Mancini

#### Date

2020-05-17

Definition in file [socket\\_wrapper.h](#).

## 7.28 socket\_wrapper.h

```
00001
00010 #ifndef SOCKET_WRAPPER_H
00011 #define SOCKET_WRAPPER_H
00012
00013 #include "logging.h"
00014 #include "network/inet_utils.h"
00015 #include "network/messages.h"
00016 #include "network/host.h"
00017
00024 class SocketWrapper{
00025 protected:
00027     struct sockaddr_in other_addr;
00028
00030     int socket_fd;
00031 public:
00035     SocketWrapper();
00036
00040     SocketWrapper(int sd) : socket_fd(sd) {}
00041
00045     int getDescriptor() {return socket_fd;};
00046
00055     Message* receiveAnyMsg(size_t size=MAX_MSG_SIZE);
00056
00068     Message* receiveMsg(MessageType type, size_t size=MAX_MSG_SIZE);
00069
00081     Message* receiveMsg(MessageType type[], int n_types, size_t size=MAX_MSG_SIZE);
00082
00089     int sendMsg(Message *msg);
00090
00097     void setOtherAddr(struct sockaddr_in addr){other_addr = addr;}
00098
00102     Host getConnectedHost() {return Host(other_addr);}
00103 };
00104
00110 class ClientSocketWrapper : public SocketWrapper{
00111 public:
00117     int connectServer(Host host);
00118 };
00119
00126 class ServerSocketWrapper : public SocketWrapper{
00127 private:
00129     struct sockaddr_in my_addr;
00130 public:
00136     ServerSocketWrapper();
00137
00143     ServerSocketWrapper(int port);
00144
00148     SocketWrapper* acceptClient();
00149
00153     int getPort() {return ntohs(my_addr.sin_port);}
00154 };
00155
00156 #endif // SOCKET_WRAPPER_H
```

## Index

- bind\_random\_port
  - inet\_utils.cpp, 26
  - inet\_utils.h, 30
- checkWin
  - Connect4, 5
- client.cpp, 15, 16
- ClientSocketWrapper, 3
  - connectServer, 3
- Connect4, 4
  - checkWin, 5
  - Connect4, 4
  - getAdv, 5
  - getNumCols, 5
  - getPlayer, 5
  - play, 6
  - print, 6
  - setPlayer, 6
- connect4.cpp, 18, 19
- connect4.h, 21, 22
- connectServer
  - ClientSocketWrapper, 3
- dump\_buffer.cpp, 22, 23
  - dump\_buffer\_hex, 23
- dump\_buffer.h, 23, 24
  - dump\_buffer\_hex, 24
- dump\_buffer\_hex
  - dump\_buffer.cpp, 23
  - dump\_buffer.h, 24
- getAdv
  - Connect4, 5
- getNumCols
  - Connect4, 5
- getPlayer
  - Connect4, 5
- Host, 7
  - Host, 7, 8
- host.h, 25
- inet\_utils.cpp, 26, 28
  - bind\_random\_port, 26
  - make\_my\_sockaddr\_in, 27
  - make\_sv\_sockaddr\_in, 27
  - sockaddr\_in\_cmp, 28
  - sockaddr\_in\_to\_string, 28
- inet\_utils.h, 29, 32
  - bind\_random\_port, 30
  - make\_my\_sockaddr\_in, 31
  - make\_sv\_sockaddr\_in, 31
  - sockaddr\_in\_cmp, 32
  - sockaddr\_in\_to\_string, 32
- logging.h, 33, 34
- make\_my\_sockaddr\_in
  - inet\_utils.cpp, 27
  - inet\_utils.h, 31
- make\_sv\_sockaddr\_in
  - inet\_utils.cpp, 27
  - inet\_utils.h, 31
- MAX\_MSG\_SIZE
  - messages.h, 37
- Message, 8
  - read, 9
  - write, 9
- messages.cpp, 35, 36
  - readMessage, 35
- messages.h, 36, 38
  - MAX\_MSG\_SIZE, 37
  - MessageType, 38
  - readMessage, 38
- MessageType
  - messages.h, 38
- MoveMessage, 9
  - read, 10
  - write, 10
- play
  - Connect4, 6
- print
  - Connect4, 6
- read
  - Message, 9
  - MoveMessage, 10
  - StartGameMessage, 15
- readMessage
  - messages.cpp, 35
  - messages.h, 38
- receiveAnyMsg
  - SocketWrapper, 12
- receiveMsg
  - SocketWrapper, 13
- sendMsg
  - SocketWrapper, 14
- ServerSocketWrapper, 10
  - ServerSocketWrapper, 11
- setOtherAddr
  - SocketWrapper, 14
- setPlayer
  - Connect4, 6
- single\_player.cpp, 39, 40
- sockaddr\_in\_cmp
  - inet\_utils.cpp, 28
  - inet\_utils.h, 32
- sockaddr\_in\_to\_string
  - inet\_utils.cpp, 28
  - inet\_utils.h, 32
- socket\_wrapper.cpp, 41



socket\_wrapper.h, [43](#), [44](#)

SocketWrapper, [12](#)

    receiveAnyMsg, [12](#)

    receiveMsg, [13](#)

    sendMsg, [14](#)

    setOtherAddr, [14](#)

StartGameMessage, [14](#)

    read, [15](#)

    write, [15](#)

write

    Message, [9](#)

    MoveMessage, [10](#)

    StartGameMessage, [15](#)