You Know UNO?

Srivatsan T
Jatin Tarachandani
Jeevan Sammeswar
Shashank Anirudh

Version V1.1
Sun Jul 25 2021

Table of Contents

CS1023-SDF-Project-Team-16 "You Know UNO?"

CS1023-SDF-Project-Team-16 created by GitHub Classroom

Building

1. Download Qt creator and clone the main branch of the repository.
2. Once the project is imported, go to Projects Tab and switch to change the RUN configurations.
3. Scroll down to Build Environment under Environment section and append the SDL library/bin directory included in the repo to the PATH.

1. Build the project using Qt 6.1.2 MinGW 64-bit compiler and CMake.
2. Find the generated executable from the build folder and make sure the dll's and sounds1 folder is present. If not add them from the libraries/bin folder.
3. Run the executable and enjoy playing!

Testing

1. Download Qt creator and clone the main branch of the repository.
2. Now try and run the cmake file from the UNO-TEST folder of the repository.
3. Run the executable and you can view the result of all the tests as output.
4. Feel free to add most tests by adding your test as a function in the private slots: tag.
5. For more details on testing check out the official Qt Documentation for the same. `https://doc.qt.io/qt-5/qtest-overview.html`

Licenses

1. The projects uses Qt6 framework and SDL libraries. Note that we use Qt's open source licensing and have thereby made our project licensed under GPL3.0 and we hereby also explicitly acknowledge the usage of Qt6 and Qt creator.
2. SDL library is available under zlib license and is also free to use. We have included the SDL license under the libraries folder.
3. We also have a bunch of .mp3 files to be used as in-game music and sound effects. All of the files are free to use and we have also included the source website and each website's licensing in the sounds1 folder as a proof that they are indeed free for use.

License Websites

1. Visit `https://www.qt.io/download-open-source?hsCtaTracking=9f6a2170-a938`

-42df-a8e2-a9f0b1d6cdce%7C6cb0de4f-9bb5-4778-ab02-bfb62735f3e5 to check Qt's documentation on using open source license.
2. Visit https://www.libsdl.org/license.php for SDL's zlib license.
3. Visit https://mixkit.co/license/#sfxFree for MixKit license.
4. Visit https://www.soundjay.com/tos.html for the offical sound-jay license.
5. Visit https://www.ashamaluevmusic.com/terms for the actual ashamaluevmusic license documentation.

Documentation

1. We used doxygen for code documentatiion and usage guide.
2. We used formatted comments in our code as well as the inbuilt documentor in doxygen to get the manual.
3. You can find it here : https://github.com/IITH-CS1023/cs1023-sdf-project-team-16/blob/main/Documentation%20of%20Code.pdf

Releases

If building projects isn't your thing you can download the latest release and extract it to find the game ready to play! Look up here for all the releases https://github.com/IITH-CS1023/cs1023-sdf-project-team-16/releases

Namespace Index

Namespace List

Here is a list of all namespaces with brief descriptions:

Hierarchical Index

Class Hierarchy

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

Class Index

Class List

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

File Index

File List

Here is a list of all files with brief descriptions:

Namespace Documentation

Ui Namespace Reference

Classes

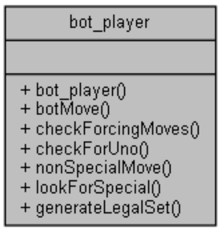
- class **greeting**

Class Documentation

bot_player Class Reference

`#include <bot_player.h>`

Collaboration diagram for bot_player:



Public Member Functions

- **bot_player** ()
- int **botMove** (**game** *g, **users** *h, QVector< **card** > *current_hand, int player_id)
- bool **checkForcingMoves** (QVector< **card** > *current_hand, **users** *h)
  Checks if there are any forced card pickups that have to be made from cards like +2 and +4.
- int **checkForUno** (QVector< **card** > *current_hand, int player_id, int number_players, **users** *h, **game** *g)
  Checks to see if the next player has one card, and looks for the strongest special card to play.
- int **nonSpecialMove** ()
  Looks for a move that isnt a special card.
- int **lookForSpecial** ()
  Looks for a special move like a +2, +4, skip, etc.
- void **generateLegalSet** (QVector< **card** > *current_hand)
  Generates the set of legal moves playable by the bot player, regenerated each turn.

Constructor & Destructor Documentation

bot_player::bot_player ()

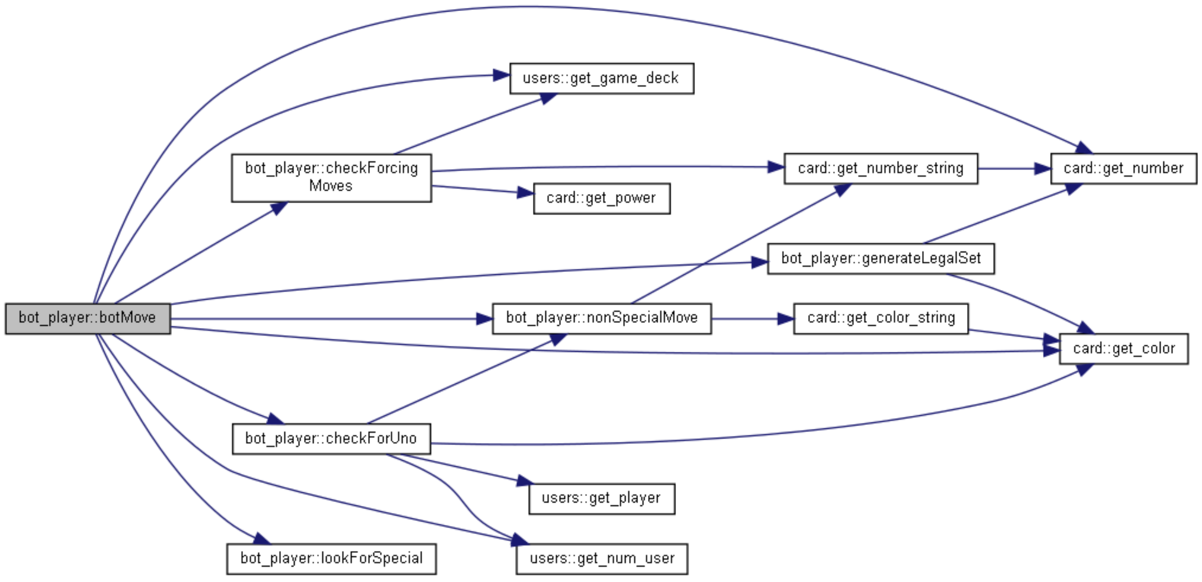
Member Function Documentation

int bot_player::botMove (game * *g*, users * *h*, QVector< card > * *current_hand*, int *player_id*)

one card added right after so this is the last position

this is to prioritise playing special moves when the card count of the hand is low to prioritise using special cards when the hand size is lower

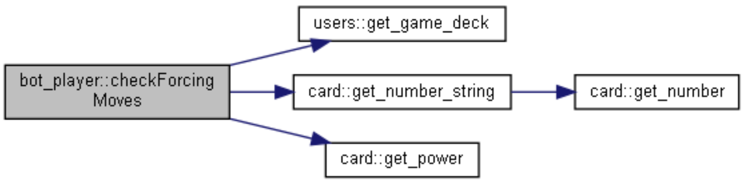
Here is the call graph for this function:



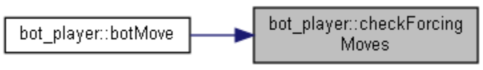
bool bot_player::checkForcingMoves (QVector< card > * *current_hand*, users * *h*)

Checks if there are any forced card pickups that have to be made from cards like +2 and +4.

Here is the call graph for this function:



Here is the caller graph for this function:

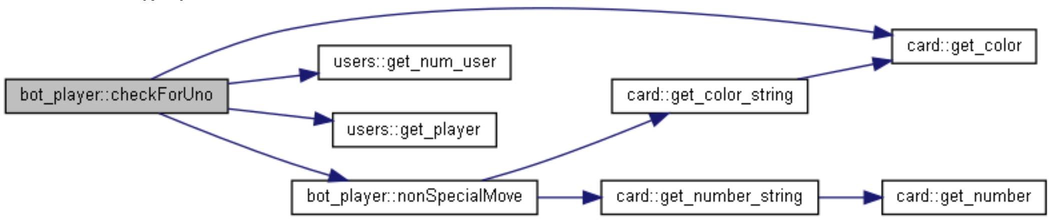


int bot_player::checkForUno (QVector< card > * *current_hand*, int *player_id*, int *number_players*, users * *h*, game * *g*)

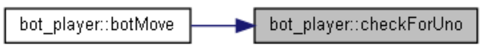
Checks to see if the next player has one card, and looks for the strongest special card to play.

if there are only 2 players then no reason to check yourself for Uno you would already be able to play or not

Here is the call graph for this function:

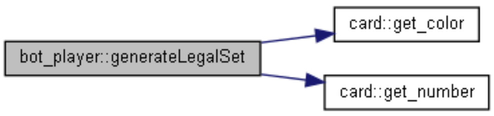


Here is the caller graph for this function:

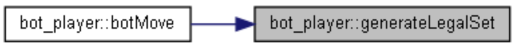


void bot_player::generateLegalSet (QVector< card > * *current_hand*)

Generates the set of legal moves playable by the bot player, regenerated each turn.
Here is the call graph for this function:

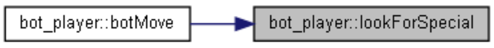


Here is the caller graph for this function:



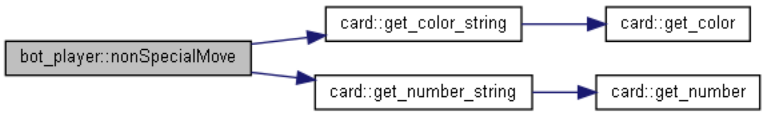
int bot_player::lookForSpecial ()

Looks for a special move like a +2, +4, skip, etc.
Here is the caller graph for this function:

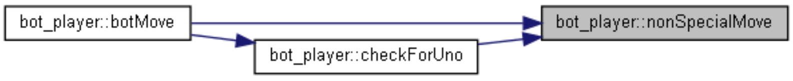


int bot_player::nonSpecialMove ()

Looks for a move that isnt a special card.
Here is the call graph for this function:



Here is the caller graph for this function:



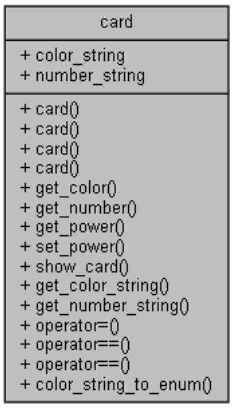
The documentation for this class was generated from the following files:

- **bot_player.h**
- **bot_player.cpp**

card Class Reference

`#include <card.h>`

Collaboration diagram for card:



Public Member Functions

- **card** (**color_t** c, **number_t** n)
  card constructor to create a card object out of given color and number

- **card** ()
- **card** (const **card** &)
- **card** (**card** *)
- **color_t get_color** () const
  A simple get function which returns the enum corresponding to the card's color.

- **number_t get_number** () const
  A simple get function which returns the enum corresponding to the card's number.

- bool **get_power** () const
  A simple get function which returns the enum corresponding to the card's power state.

- void **set_power** (bool)
  A method to change the power values of the card.

- QPushButton * **show_card** ()
  This method sets an empty button's text as the card's number and color and returns the QPushButton Object.

- QString **get_color_string** ()
  This method returns the name of the card's color as string.

- QString **get_number_string** ()
  This method returns the name of the card's number as string.

- void **operator=** (const **card** &)
- bool **operator==** (**card**) const

- bool **operator==** (**card** *) const
- void **color_string_to_enum** (QString color, QString number)
  This method is used to generate color_t and number_t enum objects from a set of strings.

Public Attributes

- QString **color_string**
  Variable to store the card's color in string.

- QString **number_string**
  Variable to store the card's number in string.

Detailed Description

Fundamental class to define a card to be used extensively in the game. Stores each card's data like its color and number and methods to create a QPushButton out of a card.

Constructor & Destructor Documentation

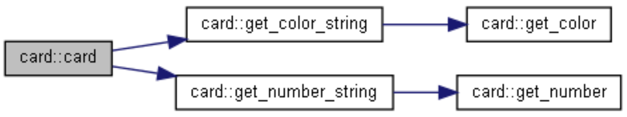
card::card (color_t *c*, number_t *n*)

card constructor to create a card object out of given color and number

Parameters

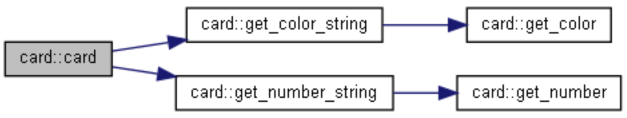
c	color_t enum object which denotes the color of the card
n	number_t enum object which denotes the number present on the card

Here is the call graph for this function:



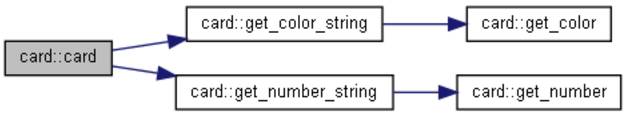
card::card ()

Here is the call graph for this function:



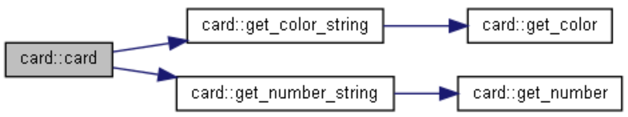
card::card (const card & *s*)

Here is the call graph for this function:



card::card (card * *s*)

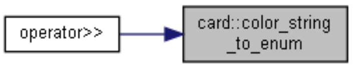
Here is the call graph for this function:



Member Function Documentation

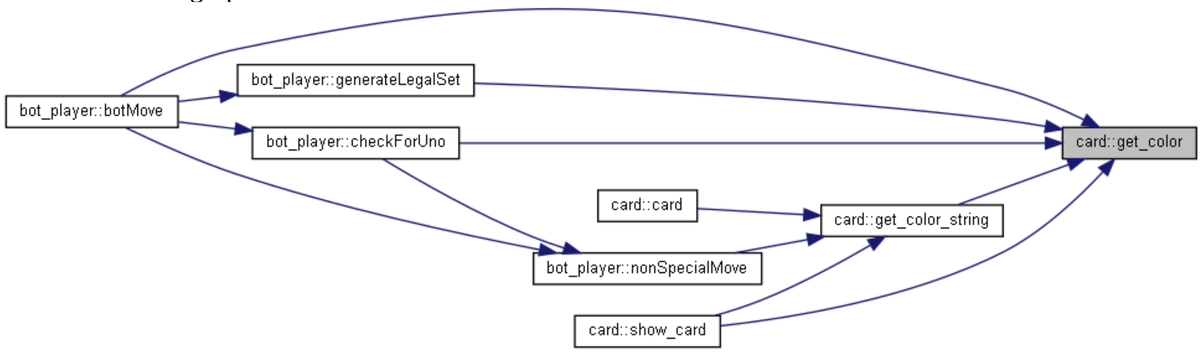
void card::color_string_to_enum (QString *color*, QString *number*)

This method is used to generate color_t and number_t enum objects from a set of strings.
Here is the caller graph for this function:



color_t card::get_color () const

A simple get function which returns the enum corresponding to the card's color.
Here is the caller graph for this function:

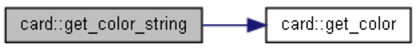


QString card::get_color_string ()

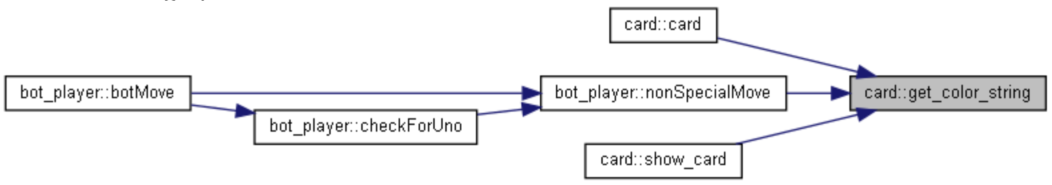
This method returns the name of the card's color as string.

This method is especially helpful in the show_card function to get the card's color as string to set button's text

Here is the call graph for this function:

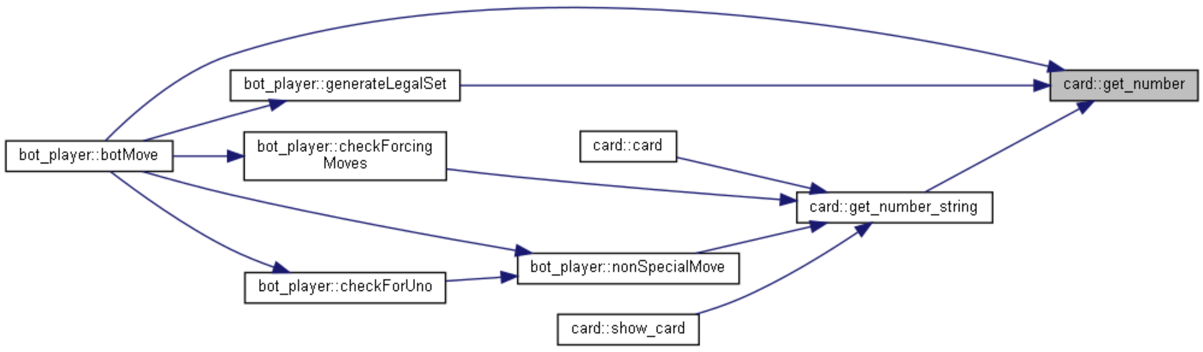


Here is the caller graph for this function:



number_t card::get_number () const

A simple get function which returns the enum corresponding to the card's number.
Here is the caller graph for this function:

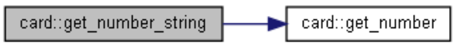


QString card::get_number_string ()

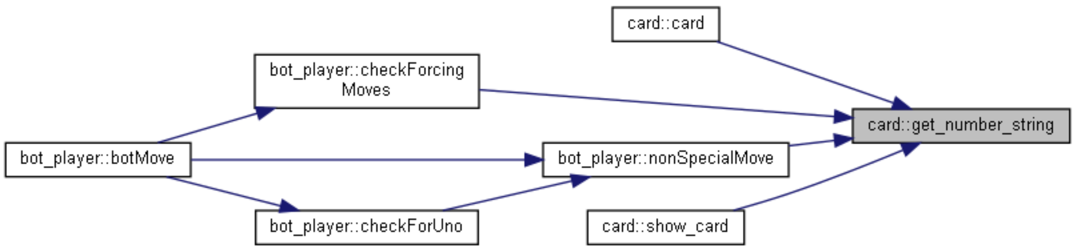
This method returns the name of the card's number as string.

This method is especially helpful in the show_card function to get the card's number as string to set button's text

Here is the call graph for this function:



Here is the caller graph for this function:

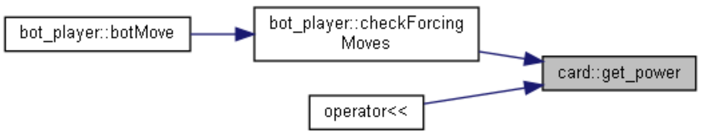


bool card::get_power () const

A simple get function which returns the enum corresponding to the card's power state.

All cards have power values of true when they are created. They lose their power once they are played by any player. This is useful in case of power up cards so that they dont affect opponents more than once!

Here is the caller graph for this function:



void card::operator= (const card & *c*)

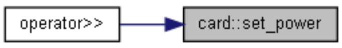
bool card::operator== (card * *c*) const

bool card::operator== (card *c*) const

void card::set_power (bool *val*)

A method to change the power values of the card.

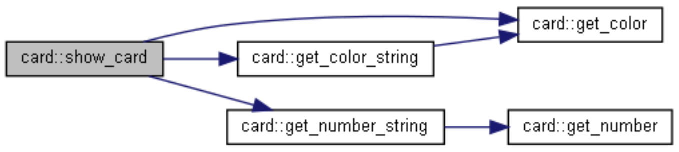
Here is the caller graph for this function:



QPushButton * card::show_card ()

This method sets an empty button's text as the card's number and color and returns the QPushButton Object.

Here is the call graph for this function:



Member Data Documentation

QString card::color_string

Variable to store the card's color in string.

QString card::number_string

Variable to store the card's number in string.

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

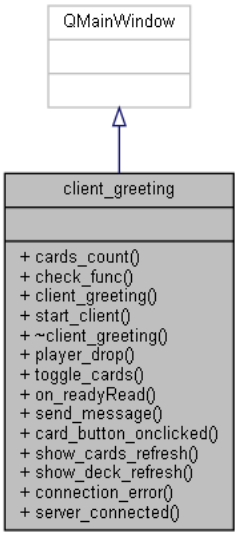
- **card.h**
- **card.cpp**

client_greeting Class Reference

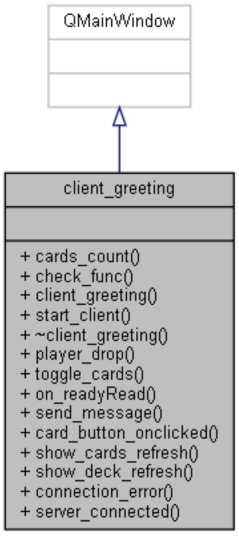
A class to handle the gameplay for client players.

`#include <client_greeting.h>`

Inheritance diagram for client_greeting:



Collaboration diagram for client_greeting:



Public Slots

- void **on_readyRead** ()
  This function is called whenever there is a new message in QDataStream. The function then decodes the message and updates its game object.

- void **send_message** (QTcpSocket *socket)
  This method serialises the user's game object and sends it via QDataStream to the QTcpSocket fed as parameter to it.*

- void **card_button_onclicked** ()

This function is called whenever a player clicks on a card button. This function then finds it's index and stores it in index variable.

- void **show_cards_refresh** ()
- void **show_deck_refresh** ()
  This function is called whenever deck_refresh_signal is emitted.

- void **connection_error** ()
- void **server_connected** ()

Signals

- void **deck_refresh_signal** ()
  This signal is emitted whenever there is a need to refresh the discard_pile alone.

- void **refresh_signal** ()

Public Member Functions

- void **cards_count** ()
  This function counts each player's cards and displays it to the given player.

- bool **check_func** ()
- **client_greeting** (unsigned, unsigned, QWidget *parent=nullptr)
- void **start_client** ()
  This function changes the title of the game window to the player's name and starts listening on the ip and port specified.

- **~client_greeting** ()
- void **player_drop** (unsigned a)
- void **toggle_cards** (bool toggle)

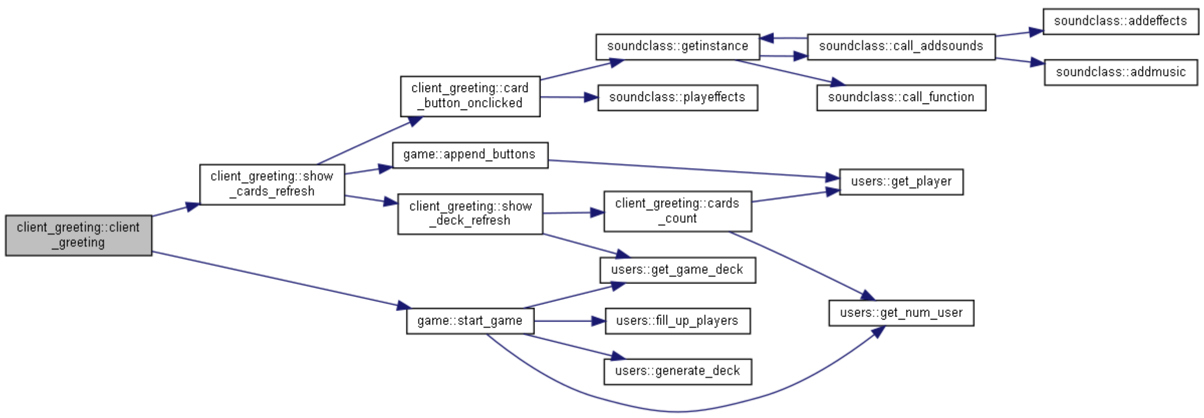
Detailed Description

A class to handle the gameplay for client players.

Constructor & Destructor Documentation

client_greeting::client_greeting (unsigned *pn*, unsigned *n*, QWidget * *parent* = `nullptr)[explicit]`

Here is the call graph for this function:



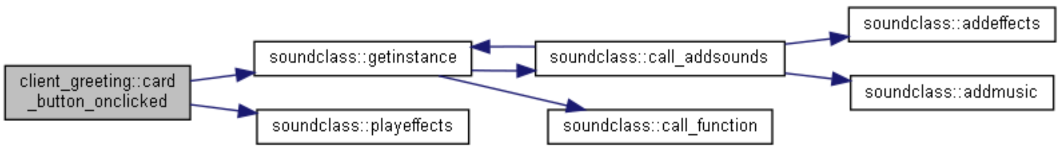
client_greeting::~client_greeting ()

Member Function Documentation

void client_greeting::card_button_onclicked () `[slot]`

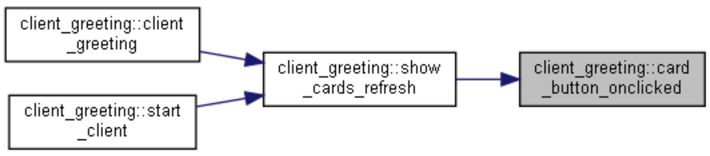
This function is called whenever a player clicks on a card button. This function then finds it's index and stores it in index variable.

Here is the call graph for this function:



client_greeting::card_button_onclicked
soundclass::getinstance
soundclass::call_addsounds
soundclass::addeffects
soundclass::addmusic
soundclass::playeffects
soundclass::call_function

Here is the caller graph for this function:

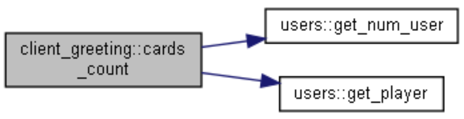


client_greeting::client_greeting
client_greeting::start_client
client_greeting::show_cards_refresh
client_greeting::card_button_onclicked

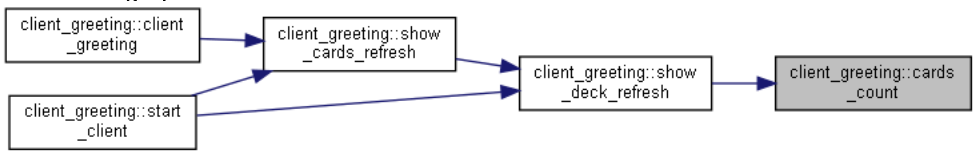
void client_greeting::cards_count ()

This function counts each player's cards and displays it to the given player.

Here is the call graph for this function:



Here is the caller graph for this function:

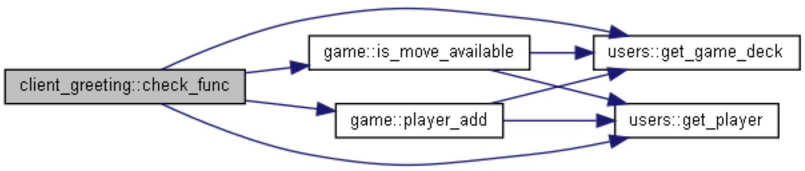


client_greeting::client_greeting
client_greeting::start_client
client_greeting::show_cards_refresh
client_greeting::show_deck_refresh
client_greeting::cards_count

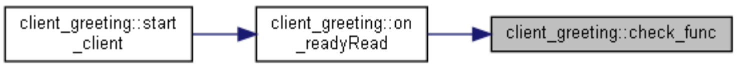
bool client_greeting::check_func ()

Check function is called whenever a player move starts to check for any pre-move conditions to implement This includes ending the move if last card was SKIP or adding cards as per power-up cards played.

Here is the call graph for this function:

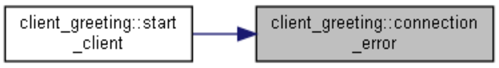


Here is the caller graph for this function:


client_greeting::start_client
client_greeting::on_readyRead
client_greeting::check_func

void client_greeting::connection_error () `[slot]`

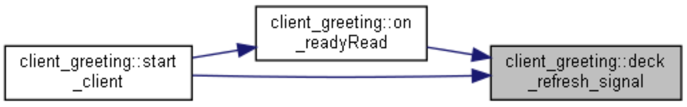
Here is the caller graph for this function:


client_greeting::start_client
client_greeting::connection_error

void client_greeting::deck_refresh_signal () `[signal]`

This signal is emitted whenever there is a need to refresh the discard_pile alone.

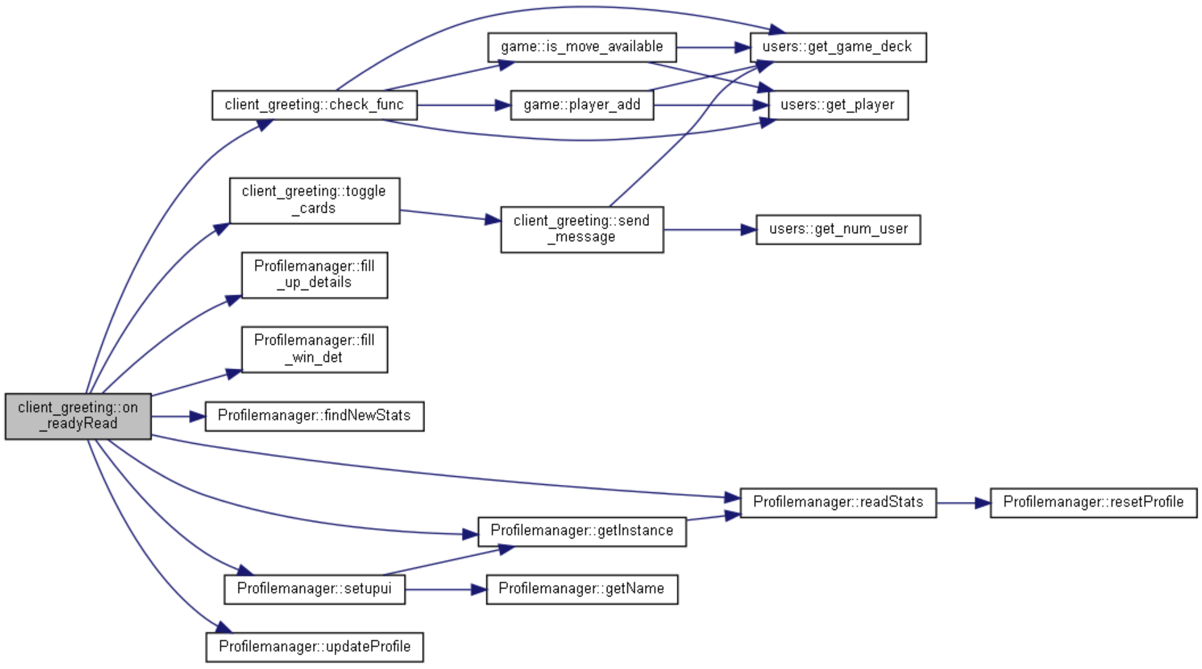
Here is the caller graph for this function:


client_greeting::start_client
client_greeting::on_readyRead
client_greeting::deck_refresh_signal

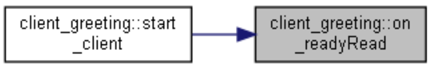
void client_greeting::on_readyRead () `[slot]`

This function is called whenever there is a new message in QDataStream. The function then decodes the message and updates its game object.

Here is the call graph for this function:


client_greeting::on_readyRead
client_greeting::check_func
game::is_move_available
game::player_add
users::get_game_deck
users::get_player
client_greeting::toggle_cards
client_greeting::send_message
users::get_num_user
Profilemanager::fill_up_details
Profilemanager::fill_win_det
Profilemanager::findNewStats
Profilemanager::readStats
Profilemanager::resetProfile
Profilemanager::getInstance
Profilemanager::setupui
Profilemanager::getName
Profilemanager::updateProfile

Here is the caller graph for this function:



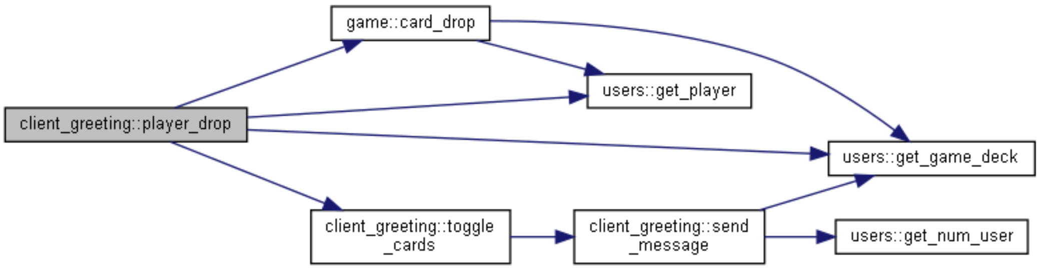
void client_greeting::player_drop (unsigned *a*)

This function uses the card_drop() from game to check if a card is valid move and drop it. This function also asks for the color of choice if it's a special card.

Parameters

a	the index of card to be dropped

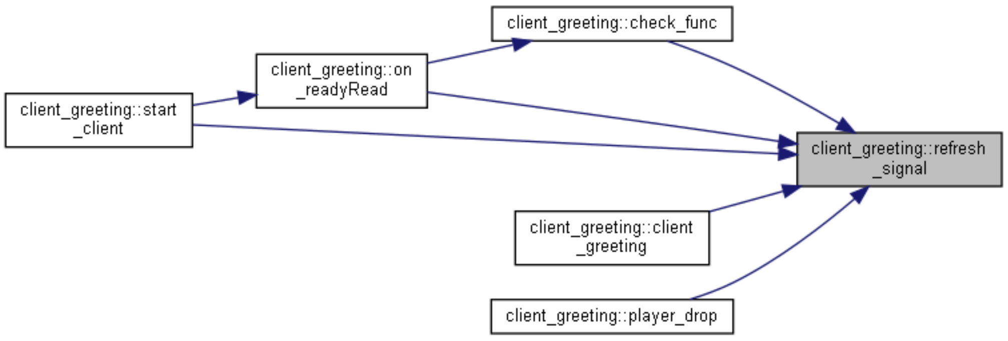
Here is the call graph for this function:


game::card_drop
users::get_player
client_greeting::player_drop
users::get_game_deck
client_greeting::toggle_cards
client_greeting::send_message
users::get_num_user

void client_greeting::refresh_signal () `[signal]`

This signal is emitted whenever we want to refresh the player's deck of cards. This method internally uses game class's append_cards() function and also calls show_deck_refresh.

Here is the caller graph for this function:


client_greeting::check_func
client_greeting::on_readyRead
client_greeting::start_client
client_greeting::refresh_signal
client_greeting::client_greeting
client_greeting::player_drop

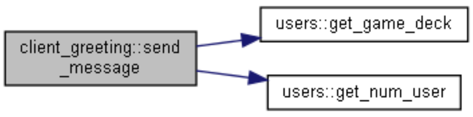
void client_greeting::send_message (QTcpSocket * *socket*) `[slot]`

This method serialises the user's game object and sends it via QDataStream to the QTcpSocket* fed as parameter to it.

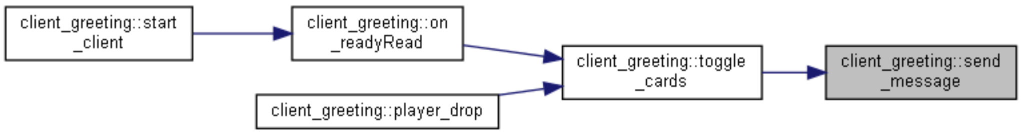
Parameters

socket	A QTcpSocket * object to denote the client socket.

Here is the call graph for this function:

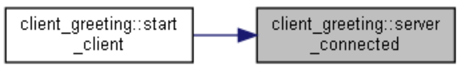


Here is the caller graph for this function:


client_greeting::start_client
client_greeting::on_readyRead
client_greeting::toggle_cards
client_greeting::send_message
client_greeting::player_drop

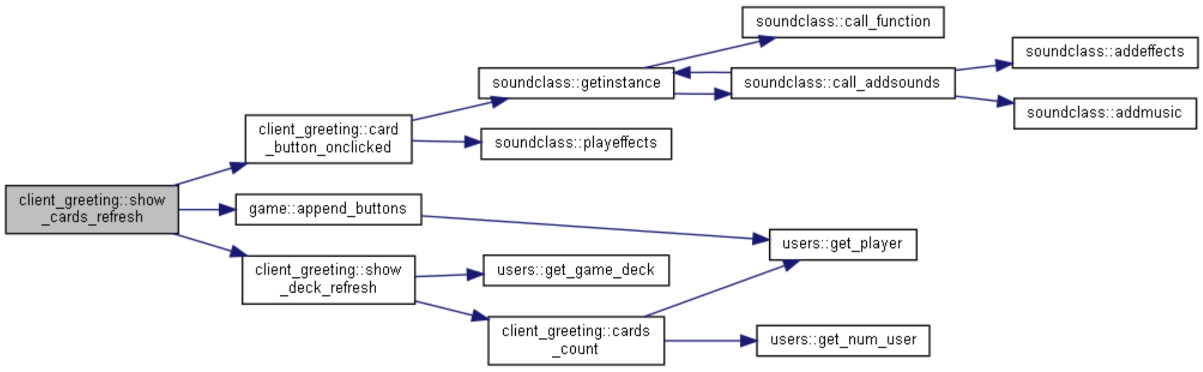
void client_greeting::server_connected () `[slot]`

Here is the caller graph for this function:

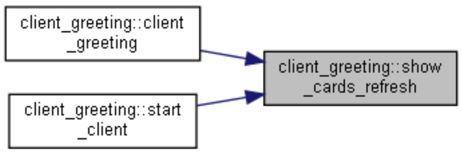

client_greeting::start_client
client_greeting::server_connected

void client_greeting::show_cards_refresh () `[slot]`

Here is the call graph for this function:


client_greeting::show_cards_refresh
client_greeting::card_button_onclicked
game::append_buttons
client_greeting::show_deck_refresh
soundclass::getinstance
soundclass::playeffects
soundclass::call_function
soundclass::call_addsounds
soundclass::addeffects
soundclass::addmusic
users::get_player
users::get_game_deck
client_greeting::cards_count
users::get_num_user

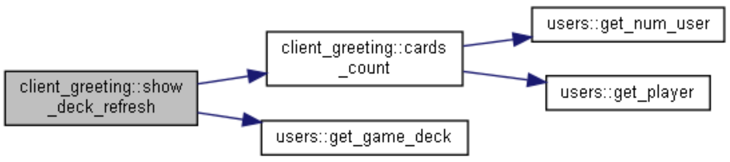
Here is the caller graph for this function:


client_greeting::client_greeting
client_greeting::start_client
client_greeting::show_cards_refresh

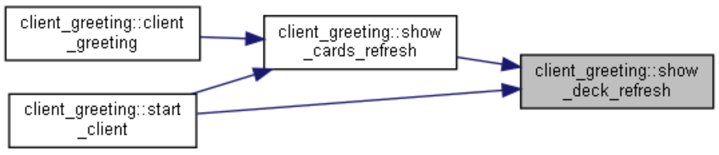
void client_greeting::show_deck_refresh () `[slot]`

This function is called whenever deck_refresh_signal is emitted.

Here is the call graph for this function:


client_greeting::show_deck_refresh
client_greeting::cards_count
users::get_game_deck
users::get_num_user
users::get_player

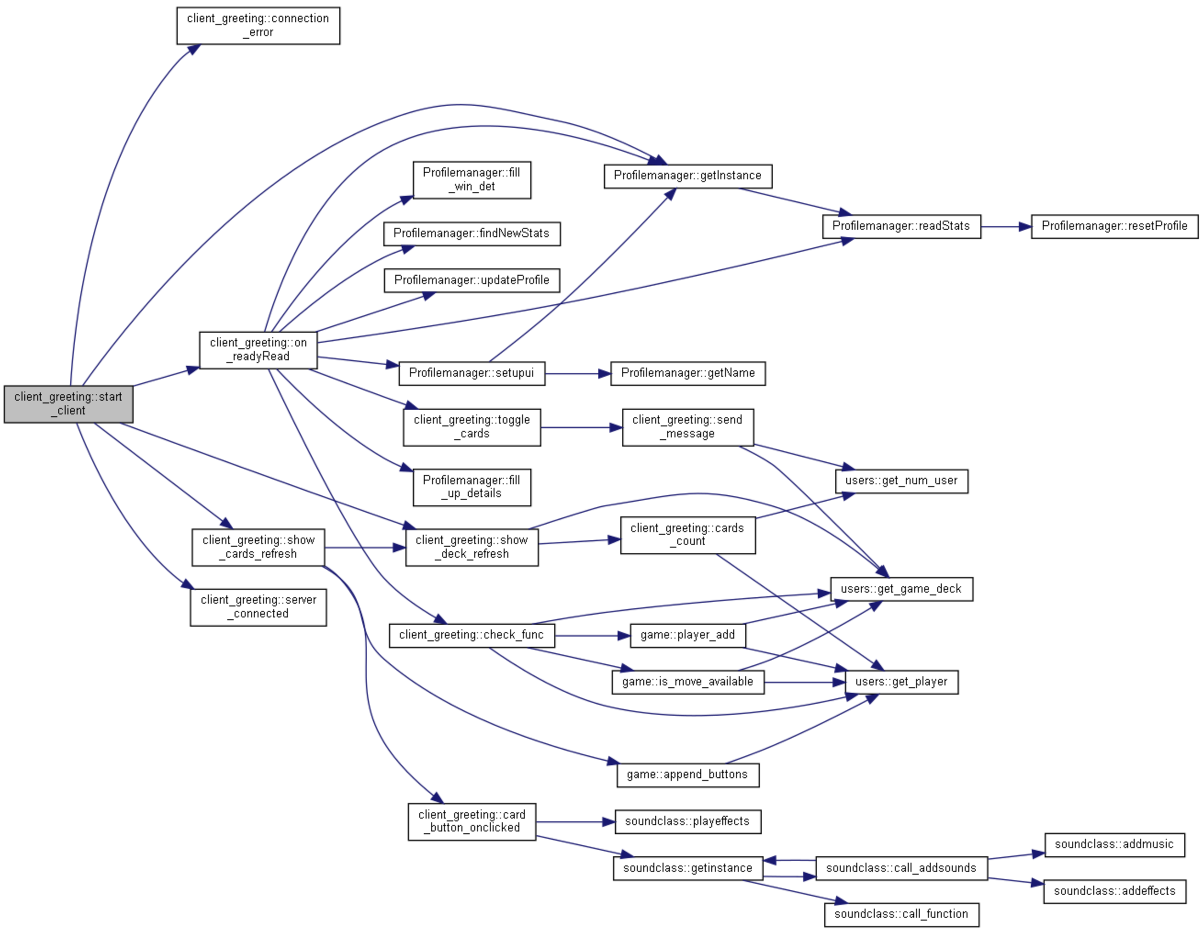
Here is the caller graph for this function:


client_greeting::client_greeting
client_greeting::start_client
client_greeting::show_cards_refresh
client_greeting::show_deck_refresh

void client_greeting::start_client ()

This function changes the title of the game window to the player's name and starts listening on the ip and port specified.

Here is the call graph for this function:


client_greeting::connection_error
Profilemanager::fill_win_det
Profilemanager::getInstance
Profilemanager::findNewStats
Profilemanager::readStats
Profilemanager::resetProfile
Profilemanager::updateProfile
client_greeting::on_readyRead
client_greeting::start_client
Profilemanager::setupui
Profilemanager::getName
client_greeting::toggle_cards
client_greeting::send_message
users::get_num_user
Profilemanager::fill_up_details
client_greeting::show_cards_refresh
client_greeting::show_deck_refresh
client_greeting::cards_count
users::get_game_deck
client_greeting::server_connected
client_greeting::check_func
game::player_add
game::is_move_available
users::get_player
game::append_buttons
client_greeting::card_button_onclicked
soundclass::playeffects
soundclass::getinstance
soundclass::call_addsounds
soundclass::addmusic
soundclass::addeffects
soundclass::call_function

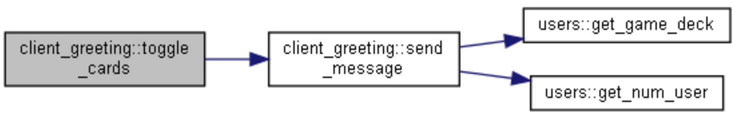
void client_greeting::toggle_cards (bool *toggle*)

This function is used to enable or disable the card buttons for the players. The buttons are disabled when other players are making their moves. The function calls send_all() method when toggled false since it denotes the move is made.

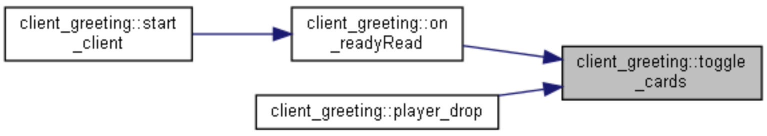
Parameters

toggle	boolean value to denote whether to toggle the button on or off

Here is the call graph for this function:



Here is the caller graph for this function:


client_greeting::start_client
client_greeting::on_readyRead
client_greeting::toggle_cards
client_greeting::player_drop

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

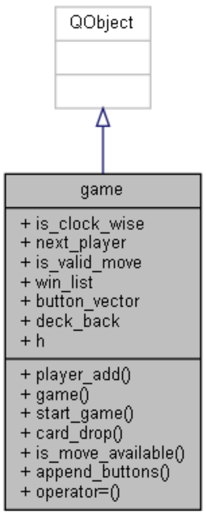
- **client_greeting.h**
- **client_greeting.cpp**

game Class Reference

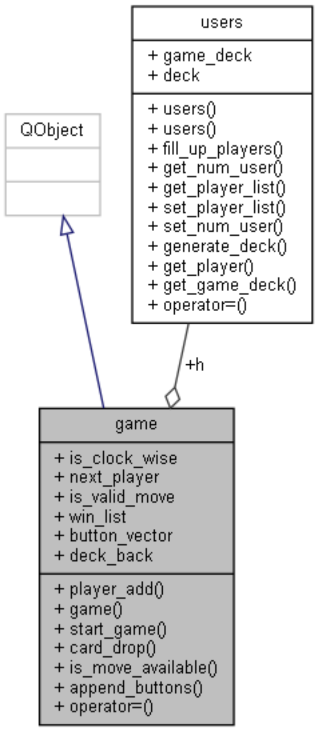
A Class to handle game variables common to both client and server.

`#include <game.h>`

Inheritance diagram for game:



Collaboration diagram for game:



Public Member Functions

- void **player_add** (unsigned)
  This function moves the card present on the top of the game deck to the back of a player's deck.
- **game** (unsigned, bool=false)
- void **start_game** ()

This function invokes the card generation function and fills up the player decks from the pool.

- bool **card_drop** (unsigned t, unsigned n)
  This function goes to the player's deck specified and check's in if the index specified is a valid move with respect to the discard pile.

- bool **is_move_available** (unsigned n)
  This function goes through the specified player's entire deck and returns if he has any valid moves to be played.

- void **append_buttons** (unsigned)
  This function clears the button vector for a player and fills it up with the card buttons returned by the get_card() function from card class.

- void **operator=** (const **game** &)
  equality operator used to compare game objects. It comes in handy since it is always getting transferred and knowing if 2 objects are same is crucial.

Public Attributes

- bool **is_clock_wise**
  Variable to store if the game_direction is clock-wise or not. Comes in handy whenver a reverse card is played and to denote the next player in queue.

- unsigned **next_player**
  Used to store the next player's number depending on is_clock_wise.

- bool **is_valid_move**
  Stores the result of ***card_drop()*** *funtion.*

- QVector< bool > **win_list**
- QVector< QPushButton * > **button_vector**
- QVector< QPushButton * > **deck_back**
- **users h**

Detailed Description

A Class to handle game variables common to both client and server.

Constructor & Destructor Documentation

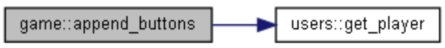
game::game (unsigned *n*, bool *color_drop* = `false`)

Member Function Documentation

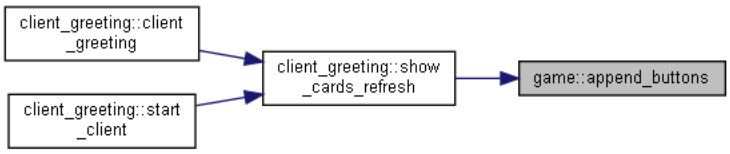
void game::append_buttons (unsigned *n*)

This function clears the button vector for a player and fills it up with the card buttons returned by the get_card() function from card class.

Here is the call graph for this function:



Here is the caller graph for this function:


client_greeting::client _greeting
client_greeting::start _client
client_greeting::show _cards_refresh
game::append_buttons

bool game::card_drop (unsigned *t*, unsigned *n*)

This function goes to the player's deck specified and check's in if the index specified is a valid move with respect to the discard pile.

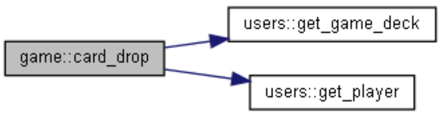
Parameters

t	To denote the player number so that the function checks his deck of cards
n	To denote the index of the card to be dropped.

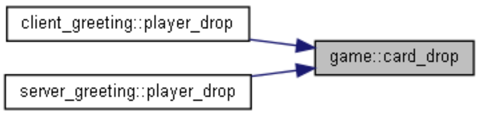
Returns

Whether the chosen index's card can be dropped or not.

Here is the call graph for this function:



Here is the caller graph for this function:



bool game::is_move_available (unsigned *n*)

This function goes through the specified player's entire deck and returns if he has any valid moves to be played.

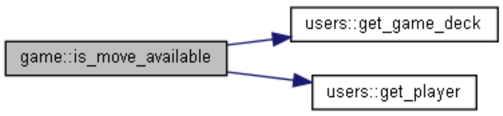
Parameters

n	Denotes the player_number

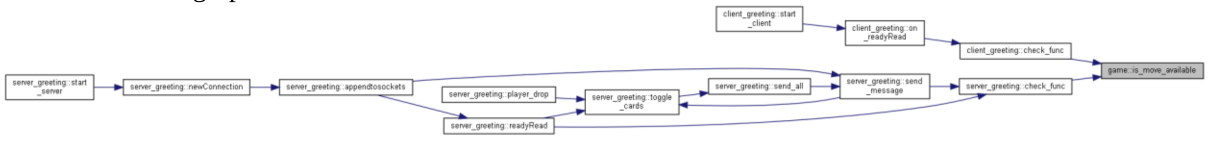
Returns

whether there is any move left to be played in n's deck.

Here is the call graph for this function:



Here is the caller graph for this function:


client_greeting::start _client
client_greeting::on _readyRead
client_greeting::check_func
game::is_move_available
server_greeting::start _server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::player_drop
server_greeting::toggle _cards
server_greeting::send_all
server_greeting::send _message
server_greeting::check_func
server_greeting::readyRead

void game::operator= (const game & *g*)

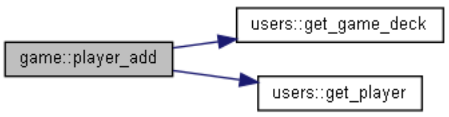
equality operator used to compare game objects. It comes in handy since it is always getting transferred and knowing if 2 objects are same is crucial.

void game::player_add (unsigned *n*)

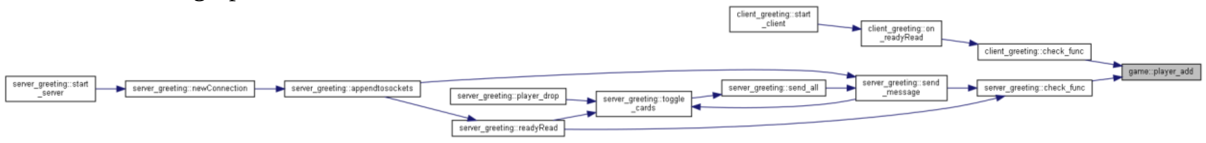
This function moves the card present on the top of the game deck to the back of a player's deck.

This function also disables the add button after adding a card to prevent multiple additions

Here is the call graph for this function:



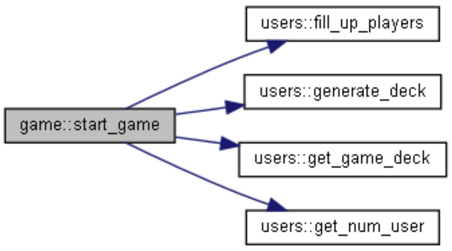
Here is the caller graph for this function:


client_greeting::start_client
client_greeting::on_readyRead
client_greeting::check_func
game::player_add
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::player_drop
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message
server_greeting::check_func
server_greeting::readyRead

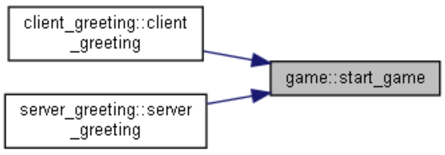
void game::start_game ()

This function invokes the card generation function and fills up the player decks from the pool.

Here is the call graph for this function:



Here is the caller graph for this function:


client_greeting::client_greeting
server_greeting::server_greeting
game::start_game

Member Data Documentation

QVector<QPushButton *> game::button_vector

QVector<QPushButton *> game::deck_back

users game::h

bool game::is_clock_wise

Variable to store if the game_direction is clock-wise or not. Comes in handy whenver a reverse card is played and to denote the next player in queue.

bool game::is_valid_move

Stores the result of **card_drop()** funtion.

unsigned game::next_player

Used to store the next player's number depending on is_clock_wise.

QVector<bool> game::win_list

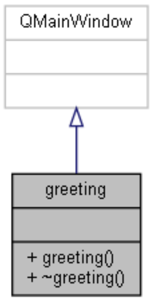
Stores the result of the game in terms of bool win values for each players. The game ends when one of these becomes true.

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

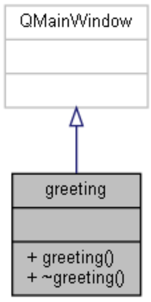
- **game.h**
- **game.cpp**

greeting Class Reference

A GUI class to display the user with navigations for start, rules, profile and exit options.
`#include <greeting.h>`
Inheritance diagram for greeting:



Collaboration diagram for greeting:



Public Member Functions

- **greeting** (QWidget *parent=nullptr)
- **~greeting** ()

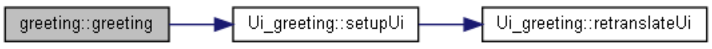
Detailed Description

A GUI class to display the user with navigations for start, rules, profile and exit options.

Constructor & Destructor Documentation

greeting::greeting (QWidget * *parent* = `nullptr`) `[explicit]`

Here is the call graph for this function:



greeting::~greeting ()

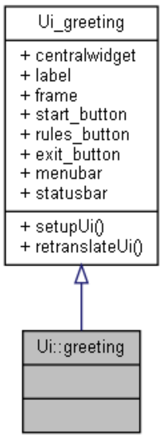
The documentation for this class was generated from the following files:

- **greeting.h**
- **greeting.cpp**

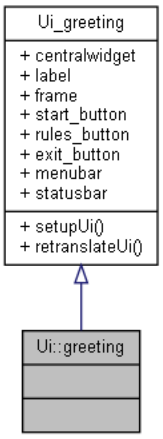
Ui::greeting Class Reference

`#include <greetings.h>`

Inheritance diagram for Ui::greeting:



Collaboration diagram for Ui::greeting:



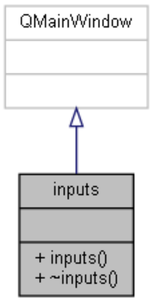
Additional Inherited Members

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

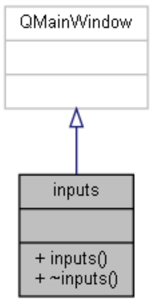
- **greetings.h**

inputs Class Reference

A GUI class to let the user enter the game settings like number of players and player_type.
`#include <inputs.h>`
Inheritance diagram for inputs:



Collaboration diagram for inputs:



Public Member Functions

- **inputs** (QWidget *parent=nullptr)
- **~inputs** ()

Detailed Description

A GUI class to let the user enter the game settings like number of players and player_type.

Constructor & Destructor Documentation

inputs::inputs (QWidget * ***parent*** **=** `nullptr`**)** `[explicit]`

inputs::~inputs ()

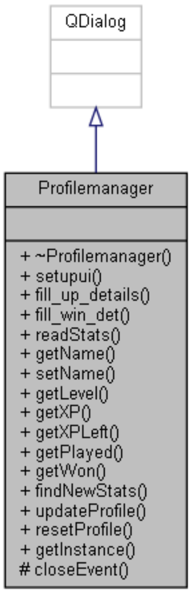
The documentation for this class was generated from the following files:

- **inputs.h**
- **inputs.cpp**

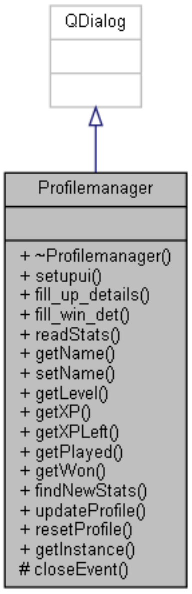
Profilemanager Class Reference

`#include <profilemanager.h>`

Inheritance diagram for Profilemanager:



Collaboration diagram for Profilemanager:



Public Member Functions

- **~Profilemanager** ()
- void **setupui** ()
- void **fill_up_details** ()
- void **fill_win_det** (bool)
- void **readStats** ()
- QString **getName** ()
- void **setName** (QString name_input)
- int **getLevel** ()

- int **getXP** ()
- int **getXPLeft** ()
- int **getPlayed** ()
- int **getWon** ()
- void **findNewStats** (bool win_status)
- void **updateProfile** ()
- void **resetProfile** ()

Static Public Member Functions

- static **Profilemanager** * **getInstance** ()

Protected Member Functions

- void **closeEvent** (QCloseEvent *event) override

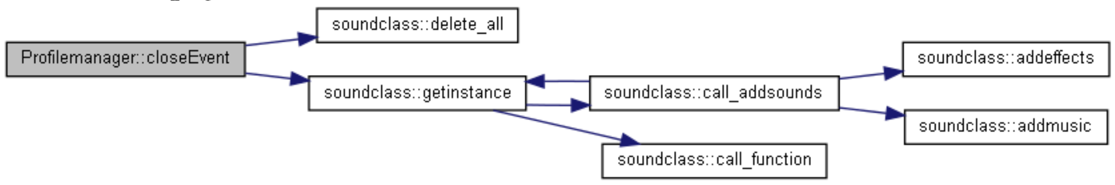
Constructor & Destructor Documentation

Profilemanager::~Profilemanager ()

Member Function Documentation

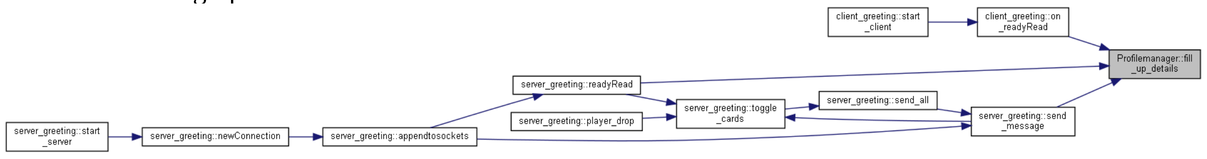
void Profilemanager::closeEvent (QCloseEvent * *event*) [override], [protected]

Here is the call graph for this function:



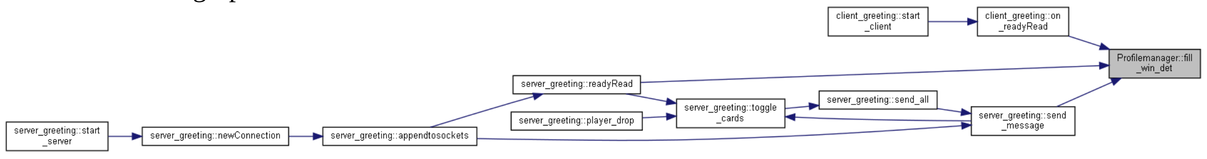
void Profilemanager::fill_up_details ()

Here is the caller graph for this function:


client_greeting::start_client
client_greeting::on_readyRead
Profilemanager::fill_up_details
server_greeting::readyRead
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::player_drop
server_greeting::send_message
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets

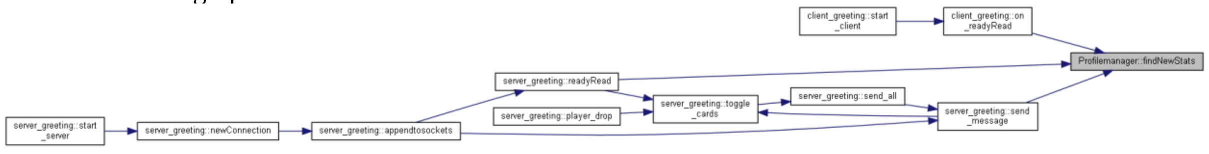
void Profilemanager::fill_win_det (bool *win_det*)

Here is the caller graph for this function:


client_greeting::start_client
client_greeting::on_readyRead
Profilemanager::fill_win_det
server_greeting::readyRead
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::player_drop
server_greeting::send_message
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets

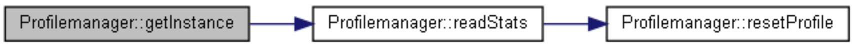
void Profilemanager::findNewStats (bool *win_status*)

Here is the caller graph for this function:

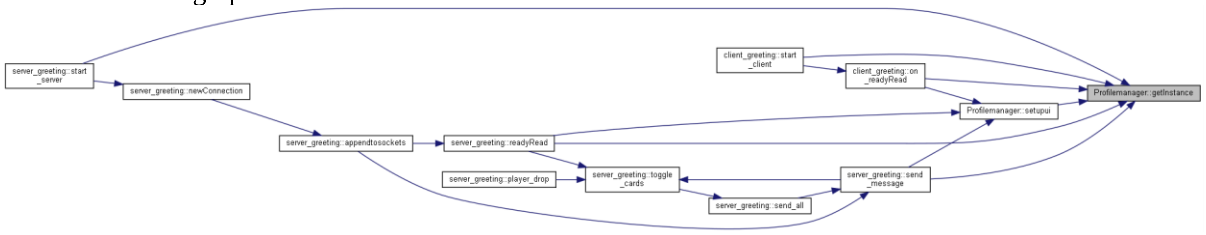

client_greeting::start_client
client_greeting::on_readyRead
Profilemanager::findNewStats
server_greeting::readyRead
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::player_drop
server_greeting::send_message
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets

Profilemanager * Profilemanager::getInstance () `[static]`

Here is the call graph for this function:

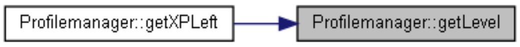

Profilemanager::getInstance
Profilemanager::readStats
Profilemanager::resetProfile

Here is the caller graph for this function:


server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead
server_greeting::player_drop
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message
client_greeting::start_client
client_greeting::on_readyRead
Profilemanager::setupui
Profilemanager::getInstance

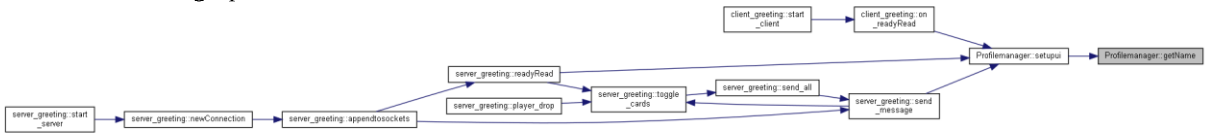
int Profilemanager::getLevel ()

Here is the caller graph for this function:



QString Profilemanager::getName ()

Here is the caller graph for this function:

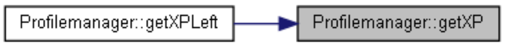

client_greeting::start_client
client_greeting::on_readyRead
server_greeting::readyRead
server_greeting::player_drop
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
Profilemanager::setupui
Profilemanager::getName

int Profilemanager::getPlayed ()

int Profilemanager::getWon ()

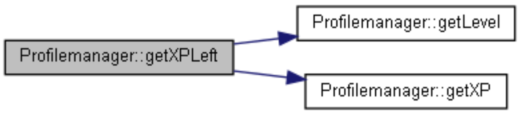
int Profilemanager::getXP ()

Here is the caller graph for this function:



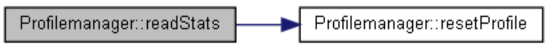
int Profilemanager::getXPLeft ()

Here is the call graph for this function:

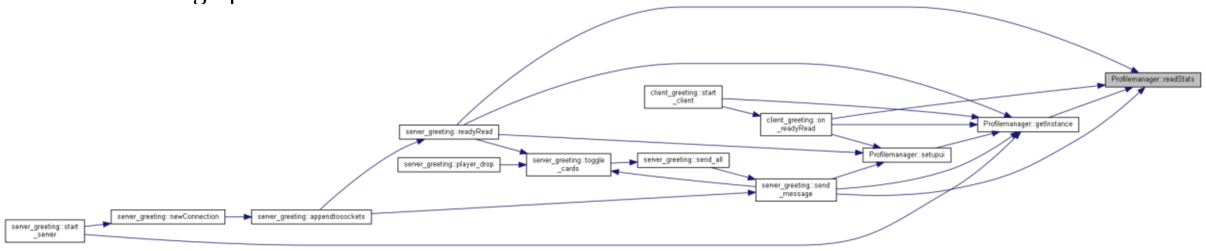


void Profilemanager::readStats ()

Here is the call graph for this function:

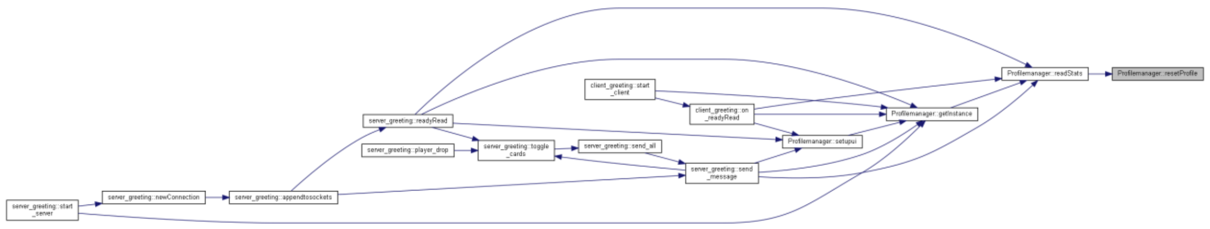


Here is the caller graph for this function:


client_greeting::start_client
client_greeting::on_readyRead
server_greeting::readyRead
server_greeting::player_drop
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
Profilemanager::setupui
Profilemanager::getInstance
Profilemanager::readStats

void Profilemanager::resetProfile ()

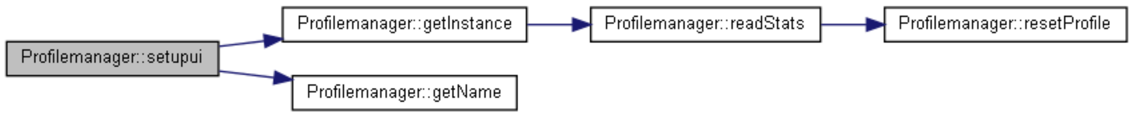
Here is the caller graph for this function:


server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead
server_greeting::player_drop
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message
client_greeting::start_client
client_greeting::on_readyRead
Profilemanager::setupui
Profilemanager::getInstance
Profilemanager::readStats
Profilemanager::resetProfile

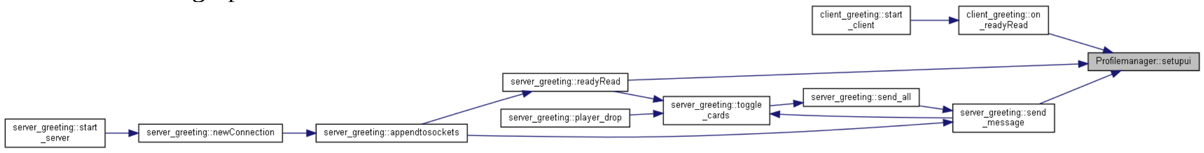
void Profilemanager::setName (QString *name_input*)

void Profilemanager::setupui ()

Here is the call graph for this function:

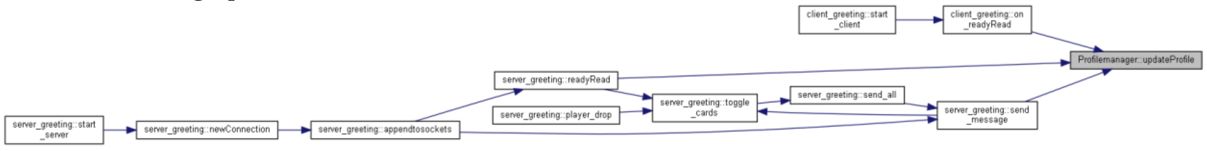

Profilemanager::setupui
Profilemanager::getInstance
Profilemanager::readStats
Profilemanager::resetProfile
Profilemanager::getName

Here is the caller graph for this function:


client_greeting::start_client
client_greeting::on_readyRead
Profilemanager::setupui
server_greeting::readyRead
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::player_drop
server_greeting::send_message
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets

void Profilemanager::updateProfile ()

Here is the caller graph for this function:

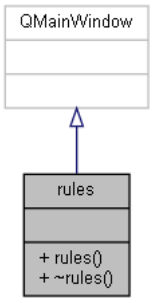

client_greeting::start_client
client_greeting::on_readyRead
Profilemanager::updateProfile
server_greeting::readyRead
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::player_drop
server_greeting::send_message
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets

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

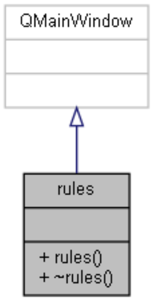
- **profilemanager.h**
- **profilemanager.cpp**

rules Class Reference

A class to display the rules of the game for the user.
`#include <rules.h>`
Inheritance diagram for rules:



Collaboration diagram for rules:



Public Member Functions

- **rules** (QWidget *parent=nullptr)
- **~rules** ()

Detailed Description

A class to display the rules of the game for the user.

Constructor & Destructor Documentation

rules::rules (QWidget * *parent* = `nullptr`) `[explicit]`

rules::~rules ()

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

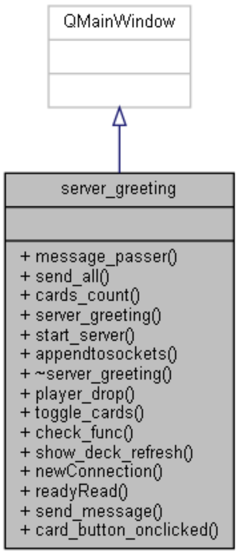
- **rules.h**
- **rules.cpp**

server_greeting Class Reference

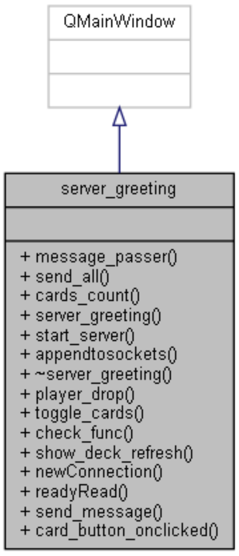
A class to handle the gameplay for the server player.

```
#include <server_greeting.h>
```

Inheritance diagram for server_greeting:



Collaboration diagram for server_greeting:



Public Slots

- void **show_deck_refresh** ()
  This function is called whenever deck_refresh_signal is emitted.

- void **newConnection** ()
  *This function is called whenever a new client connects to the server. This function internally calls **appendtosockets(QTcpSocket*)** function.*

- void **readyRead** ()

This function is called whenever there is a new message in QDataStream. The function then decodes the message and updates its game object.

- void **send_message** (QTcpSocket *)

  This method serialises the user's game object and sends it via QDataStream to the QTcpSocket fed as parameter to it.*

- void **card_button_onclicked** ()

  This function is called whenever a player clicks on a card button. This function then finds it's index and stores it in index variable.

Signals

- void **refresh_signal** ()
- void **deck_refresh_signal** ()

  This signal is emitted whenever there is a need to refresh the discard_pile alone.

Public Member Functions

- void **message_passer** ()
- void **send_all** ()

  This function is used by the server to send the modified game object to all clients after making its move.

- void **cards_count** ()

  This function counts each player's cards and displays it to the given player.

- **server_greeting** (unsigned, bool, QWidget *parent=nullptr)
- void **start_server** ()

  This function changes the title of the window to given player's name from profilemanager and starts listening for potential clients.

- void **appendtosockets** (QTcpSocket *socket)

  This function adds the clients to a vector of clients called sockets and enables start_game button if all clients have joined.

- **~server_greeting** ()
- void **player_drop** (unsigned)
- void **toggle_cards** (bool)
- bool **check_func** ()

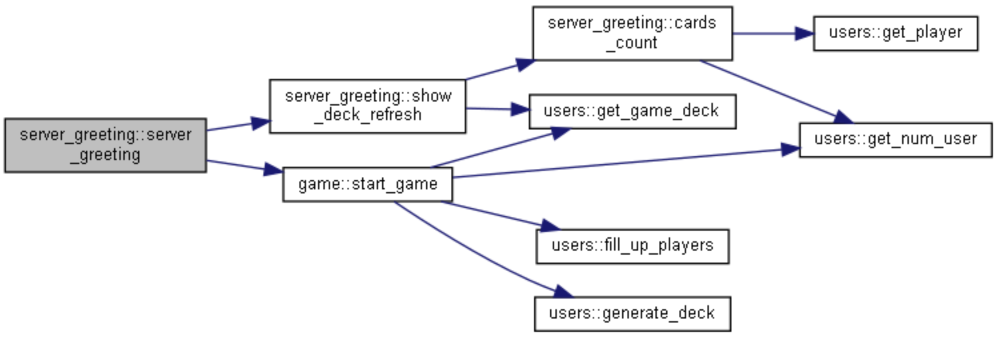
Detailed Description

A class to handle the gameplay for the server player.

Constructor & Destructor Documentation

server_greeting::server_greeting (unsigned *n*, bool *card_drop*, QWidget * *parent* = `nullptr`) `[explicit]`

Here is the call graph for this function:



server_greeting::~server_greeting ()

Member Function Documentation

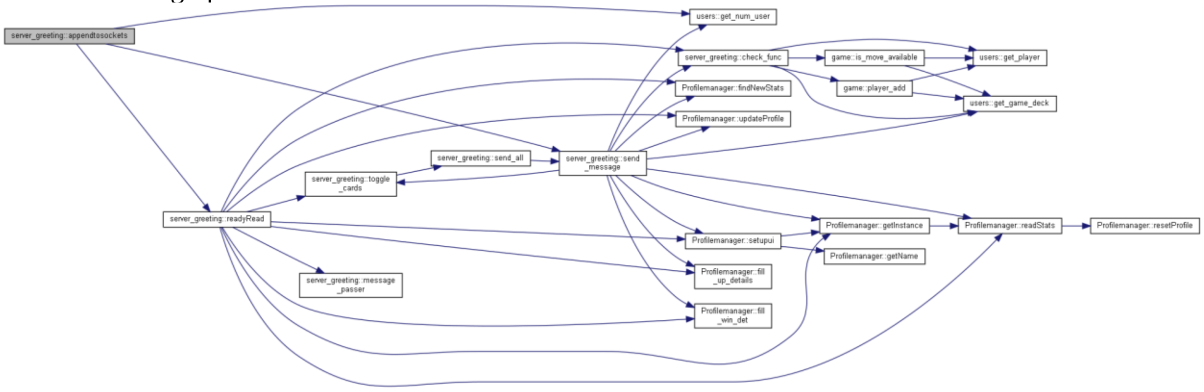
void server_greeting::appendtosockets (QTcpSocket * *socket*)

This function adds the clients to a vector of clients called sockets and enables start_game button if all clients have joined.

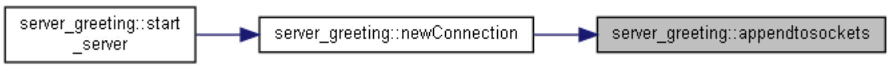
Parameters

socket	A QTcpSocket* object to denote the client socket.

Here is the call graph for this function:


server_greeting::appendtosockets
users::get_num_user
server_greeting::check_func
game::is_move_available
users::get_player
game::player_add
users::get_game_deck
Profilemanager::findNewStats
Profilemanager::updateProfile
server_greeting::send_all
server_greeting::send _message
server_greeting::toggle _cards
server_greeting::readyRead
Profilemanager::getInstance
Profilemanager::readStats
Profilemanager::resetProfile
Profilemanager::setupui
Profilemanager::getName
server_greeting::message _passer
Profilemanager::fill _up_details
Profilemanager::fill _win_det

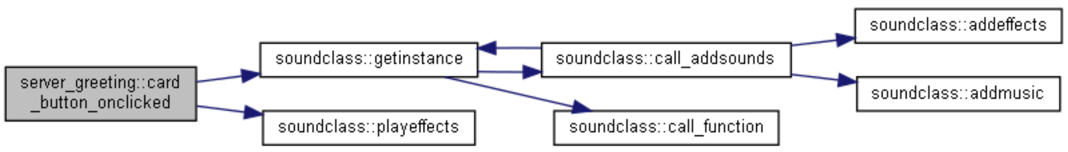
Here is the caller graph for this function:


server_greeting::start _server
server_greeting::newConnection
server_greeting::appendtosockets

void server_greeting::card_button_onclicked () `[slot]`

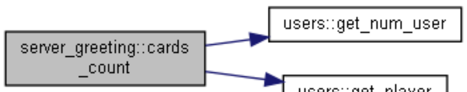
This function is called whenever a player clicks on a card button. This function then finds it's index and stores it in index variable.

Here is the call graph for this function:

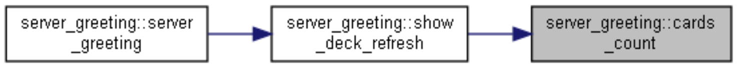

server_greeting::card_button_onclicked
soundclass::getinstance
soundclass::playeffects
soundclass::call_addsounds
soundclass::call_function
soundclass::addeffects
soundclass::addmusic

void server_greeting::cards_count ()

This function counts each player's cards and displays it to the given player.
Here is the call graph for this function:

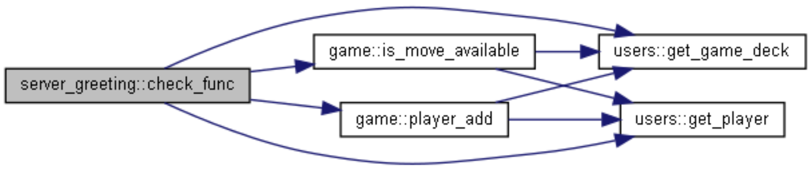


Here is the caller graph for this function:

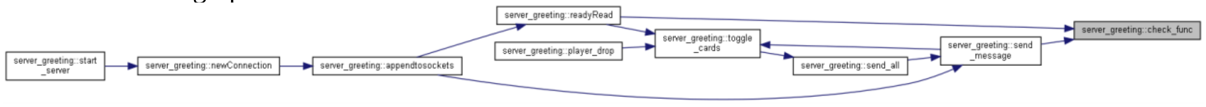

server_greeting::server_greeting
server_greeting::show_deck_refresh
server_greeting::cards_count

bool server_greeting::check_func ()

Check function is called whenever a player move starts to check for any pre-move conditions to implement This includes ending the move if last card was SKIP or adding cards as per power-up cards played.
Here is the call graph for this function:

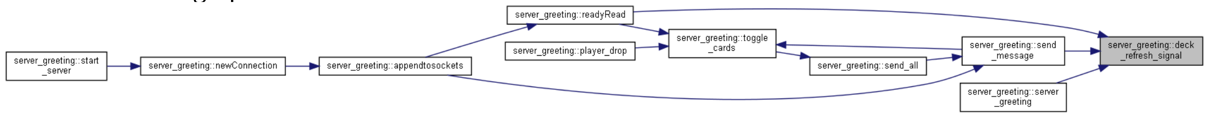


Here is the caller graph for this function:


server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead
server_greeting::player_drop
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message
server_greeting::check_func

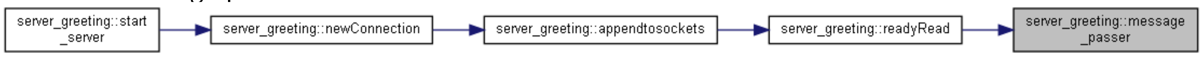
void server_greeting::deck_refresh_signal () `[signal]`

This signal is emitted whenever there is a need to refresh the discard_pile alone.
Here is the caller graph for this function:


server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead
server_greeting::player_drop
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message
server_greeting::server_greeting
server_greeting::deck_refresh_signal

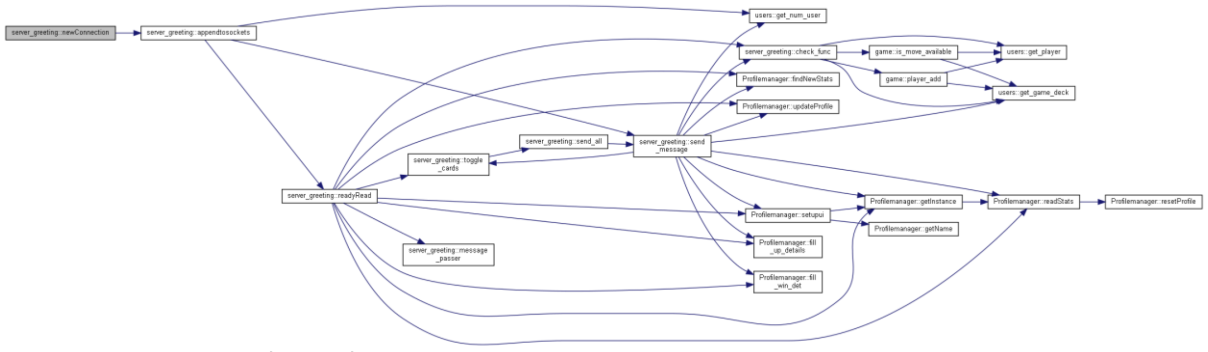
void server_greeting::message_passer ()

This function is used to relay the modified game object that was sent to server by a client to all the other clients as that they can update their game objects.
Here is the caller graph for this function:

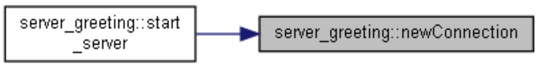

server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead
server_greeting::message_passer

void server_greeting::newConnection () `[slot]`

This function is called whenever a new client connects to the server. This function internally calls **appendtosockets(QTcpSocket*)** function.
Here is the call graph for this function:


server_greeting::newConnection
server_greeting::appendtosockets
users::get_num_user
server_greeting::check_func
game::is_move_available
users::get_player
Profilemanager::findNewStats
game::player_add
users::get_game_deck
Profilemanager::updateProfile
server_greeting::send_all
server_greeting::send _message
server_greeting::toggle _cards
server_greeting::readyRead
Profilemanager::getInstance
Profilemanager::readStats
Profilemanager::resetProfile
Profilemanager::setupui
Profilemanager::getName
Profilemanager::fill _up_details
server_greeting::message _passer
Profilemanager::fill _win_det

Here is the caller graph for this function:


server_greeting::start _server
server_greeting::newConnection

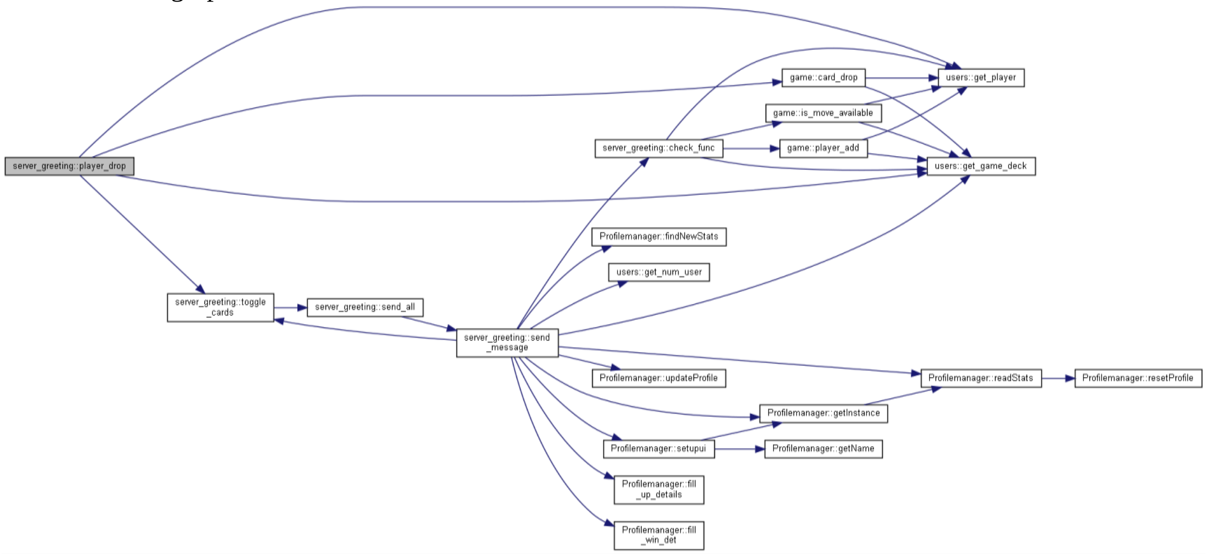
void server_greeting::player_drop (unsigned *n*)

This function uses the card_drop() from game to check if a card is valid move and drop it. This function also asks for the color of choice if it's a special card.

Parameters

toggle	boolean value to denote whether to toggle the button on or off

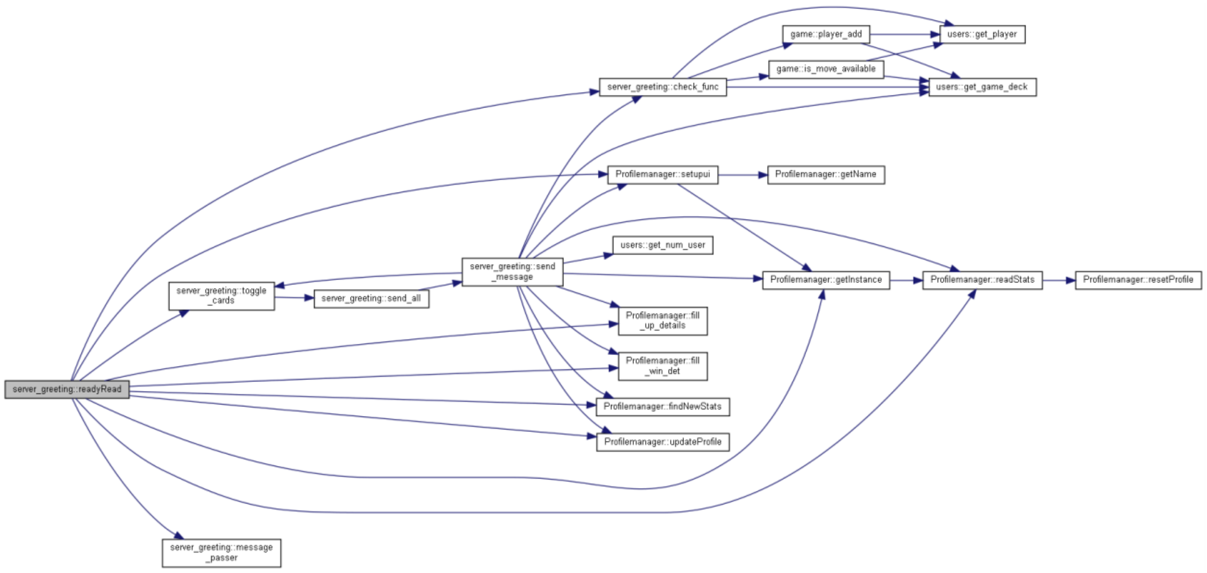
Here is the call graph for this function:


server_greeting::player_drop
game::card_drop
users::get_player
game::is_move_available
server_greeting::check_func
game::player_add
users::get_game_deck
Profilemanager::findNewStats
users::get_num_user
server_greeting::toggle _cards
server_greeting::send_all
server_greeting::send _message
Profilemanager::updateProfile
Profilemanager::readStats
Profilemanager::resetProfile
Profilemanager::getInstance
Profilemanager::setupui
Profilemanager::getName
Profilemanager::fill _up_details
Profilemanager::fill _win_det

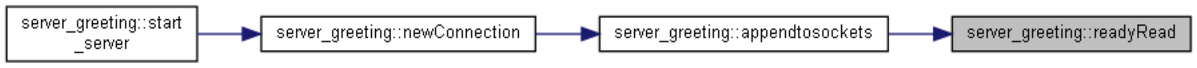
void server_greeting::readyRead () `[slot]`

This function is called whenever there is a new message in QDataStream. The function then decodes the message and updates its game object.

Here is the call graph for this function:


game::player_add
users::get_player
game::is_move_available
server_greeting::check_func
users::get_game_deck
Profilemanager::setupui
Profilemanager::getName
users::get_num_user
server_greeting::send_message
server_greeting::toggle_cards
server_greeting::send_all
Profilemanager::getInstance
Profilemanager::readStats
Profilemanager::resetProfile
Profilemanager::fill_up_details
Profilemanager::fill_win_det
server_greeting::readyRead
Profilemanager::findNewStats
Profilemanager::updateProfile
server_greeting::message_passer

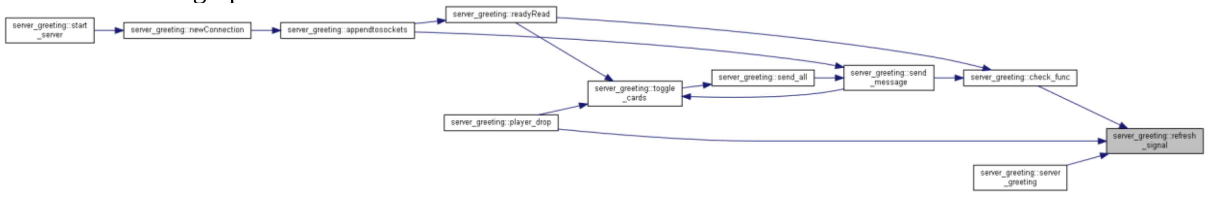
Here is the caller graph for this function:


server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead

void server_greeting::refresh_signal () `[signal]`

This signal is emitted whenever we want to refresh the player's deck of cards. This method internally uses game class's append_cards() function and also calls show_deck_refresh.

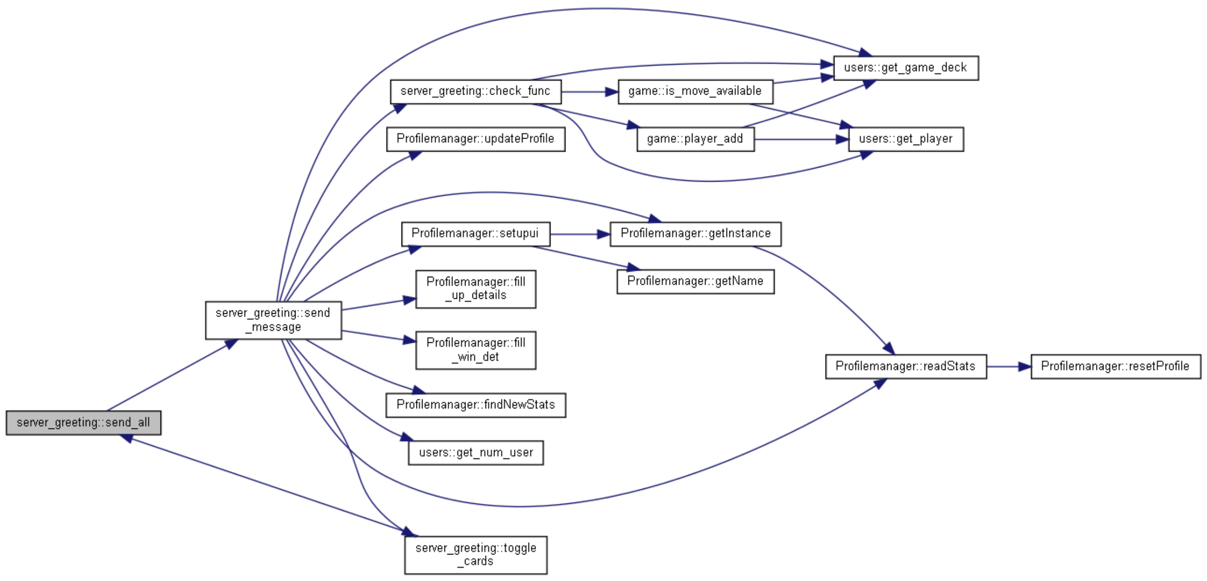
Here is the caller graph for this function:


server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead
server_greeting::send_all
server_greeting::send_message
server_greeting::check_func
server_greeting::toggle_cards
server_greeting::player_drop
server_greeting::refresh_signal
server_greeting::server_greeting

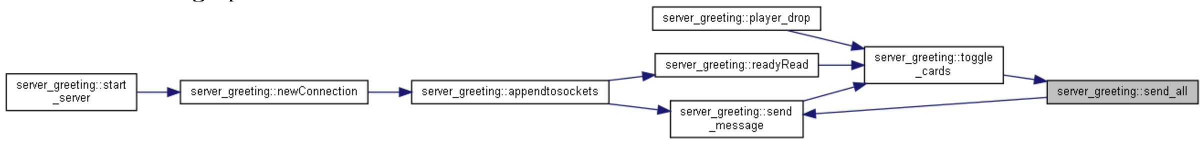
void server_greeting::send_all ()

This function is used by the server to send the modified game object to all clients after making its move.

Here is the call graph for this function:


users::get_game_deck
server_greeting::check_func
game::is_move_available
Profilemanager::updateProfile
game::player_add
users::get_player
Profilemanager::setupui
Profilemanager::getInstance
Profilemanager::fill_up_details
Profilemanager::getName
server_greeting::send_message
Profilemanager::fill_win_det
Profilemanager::readStats
Profilemanager::resetProfile
Profilemanager::findNewStats
server_greeting::send_all
users::get_num_user
server_greeting::toggle_cards

Here is the caller graph for this function:


server_greeting::player_drop
server_greeting::toggle_cards
server_greeting::readyRead
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::send_all
server_greeting::send_message

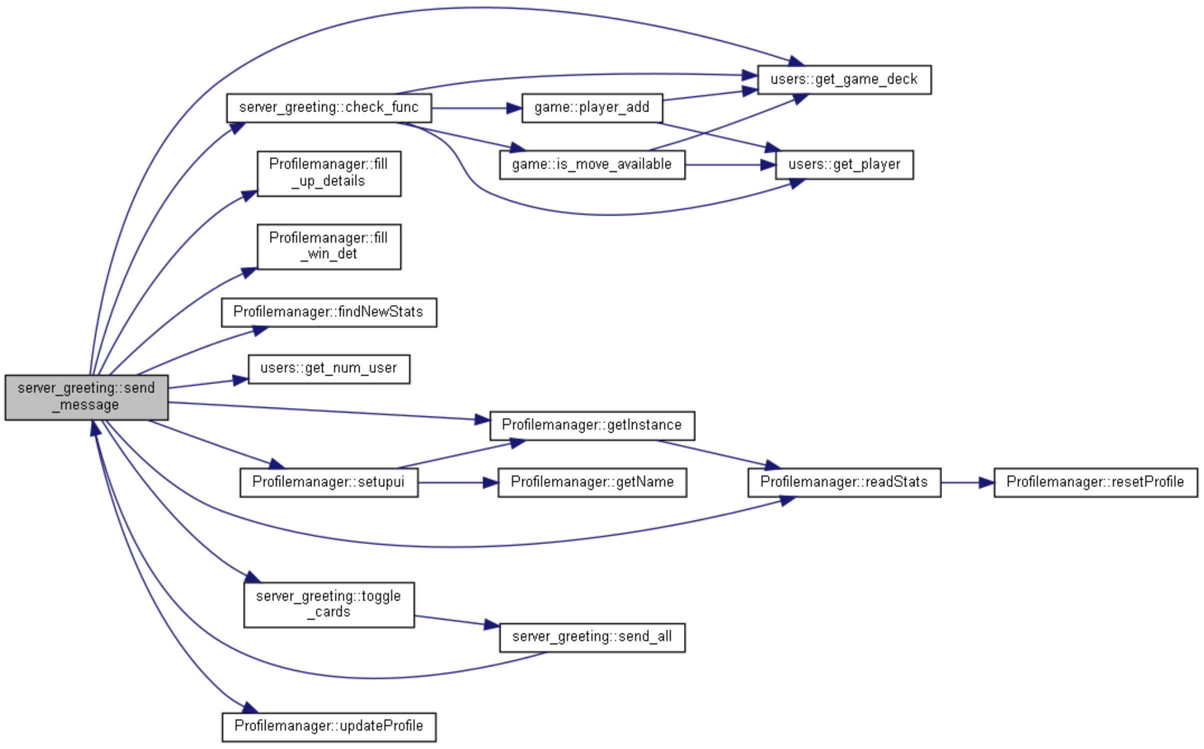
void server_greeting::send_message (QTcpSocket * *c*)`[slot]`

This method serialises the user's game object and sends it via QDataStream to the QTcpSocket* fed as parameter to it.

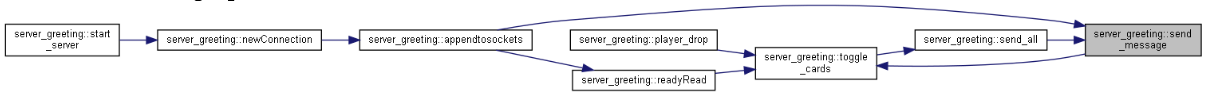
Parameters

socket	A QTcpSocket * object to denote the client socket.

Here is the call graph for this function:


users::get_game_deck
server_greeting::check_func
game::player_add
Profilemanager::fill_up_details
game::is_move_available
users::get_player
Profilemanager::fill_win_det
Profilemanager::findNewStats
users::get_num_user
server_greeting::send_message
Profilemanager::getInstance
Profilemanager::setupui
Profilemanager::getName
Profilemanager::readStats
Profilemanager::resetProfile
server_greeting::toggle_cards
server_greeting::send_all
Profilemanager::updateProfile

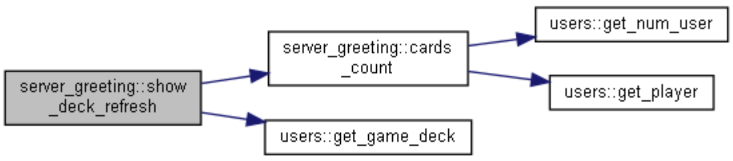
Here is the caller graph for this function:


server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::player_drop
server_greeting::readyRead
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message

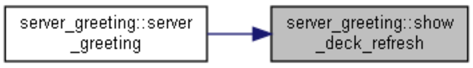
void server_greeting::show_deck_refresh () `[slot]`

This function is called whenever deck_refresh_signal is emitted.

Here is the call graph for this function:


server_greeting::show_deck_refresh
server_greeting::cards_count
users::get_num_user
users::get_player
users::get_game_deck

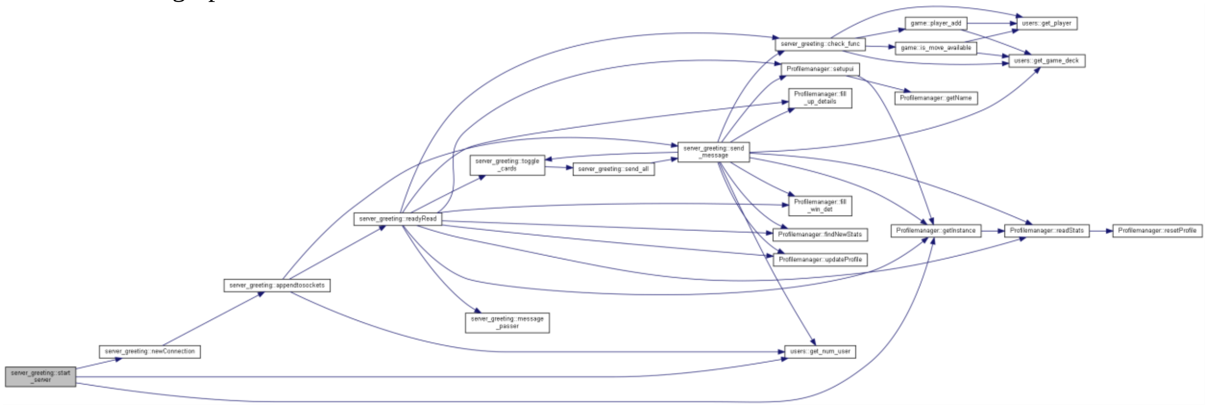
Here is the caller graph for this function:


server_greeting::server_greeting
server_greeting::show_deck_refresh

void server_greeting::start_server ()

This function changes the title of the window to given player's name from profilemanager and starts listening for potential clients.

Here is the call graph for this function:


server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message
server_greeting::message_passer
server_greeting::check_func
game::player_add
game::is_move_available
users::get_player
users::get_game_deck
Profilemanager::setupui
Profilemanager::getName
Profilemanager::fill_up_details
Profilemanager::fill_win_det
Profilemanager::findNewStats
Profilemanager::updateProfile
Profilemanager::getInstance
Profilemanager::readStats
Profilemanager::resetProfile
users::get_num_user

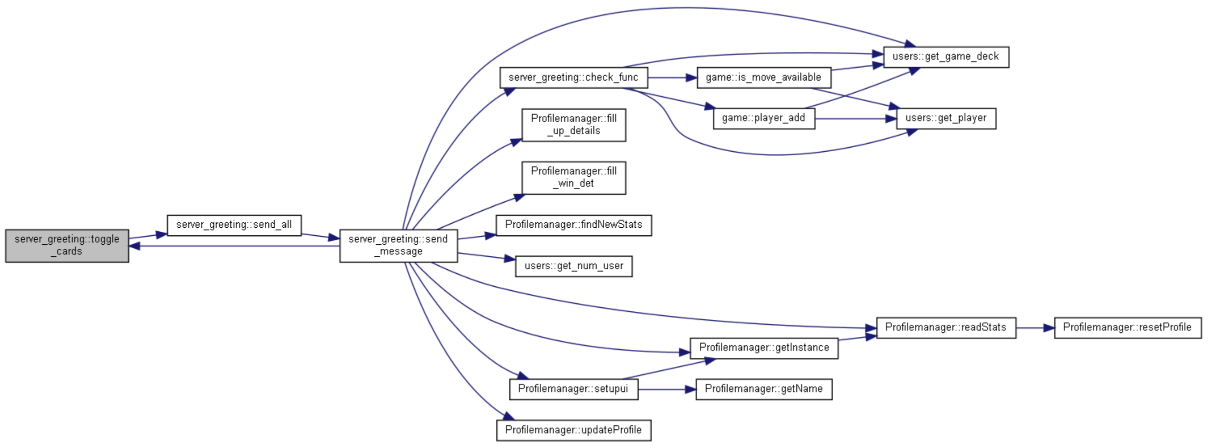
void server_greeting::toggle_cards (bool *choice*)

This function is used to enable or disable the card buttons for the players. The buttons are disabled when other players are making their moves. The function calls **send_all()** method when toggled false since it denotes the move is made.

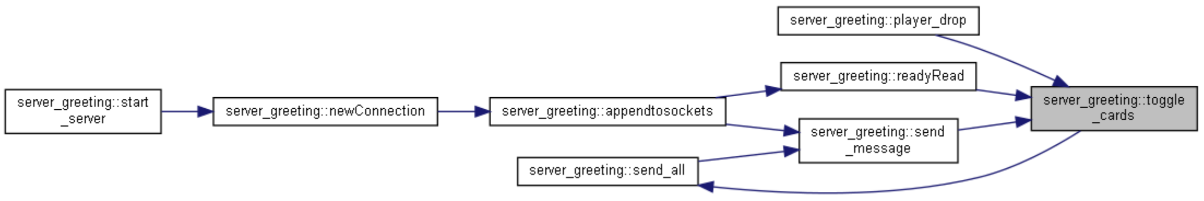
Parameters

toggle	boolean value to denote whether to toggle the button on or off

Here is the call graph for this function:


server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message
server_greeting::check_func
game::is_move_available
users::get_game_deck
game::player_add
users::get_player
Profilemanager::fill_up_details
Profilemanager::fill_win_det
Profilemanager::findNewStats
users::get_num_user
Profilemanager::readStats
Profilemanager::resetProfile
Profilemanager::getInstance
Profilemanager::setupui
Profilemanager::getName
Profilemanager::updateProfile

Here is the caller graph for this function:


server_greeting::player_drop
server_greeting::readyRead
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::send_message
server_greeting::toggle_cards
server_greeting::send_all

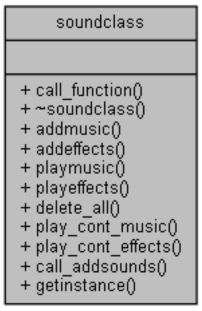
The documentation for this class was generated from the following files:

- **server_greeting.h**
- **server_greeting.cpp**

soundclass Class Reference

`#include <soundclass.h>`

Collaboration diagram for soundclass:



Public Member Functions

- void **call_function** ()
- **~soundclass** ()
- void **addmusic** (const char *path)
- void **addeffects** (const char *path)
- void **playmusic** (int num) const
- void **playeffects** (int num) const
- void **delete_all** ()
- void **play_cont_music** (int num)
- void **play_cont_effects** (int num)
- void **call_addsounds** ()

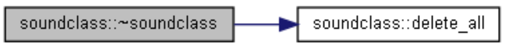
Static Public Member Functions

- static **soundclass** * **getinstance** ()

Constructor & Destructor Documentation

soundclass::~soundclass ()

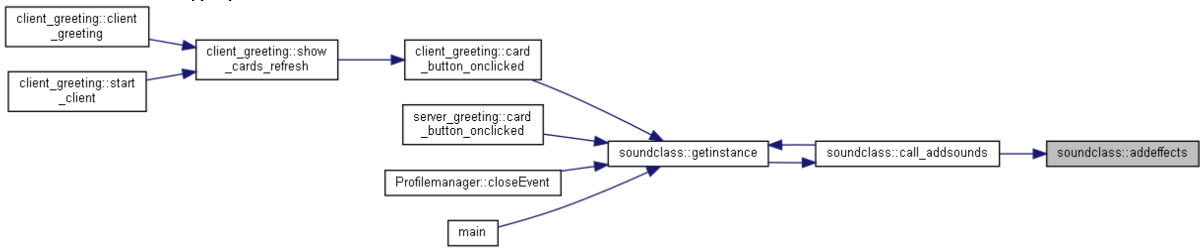
Here is the call graph for this function:



Member Function Documentation

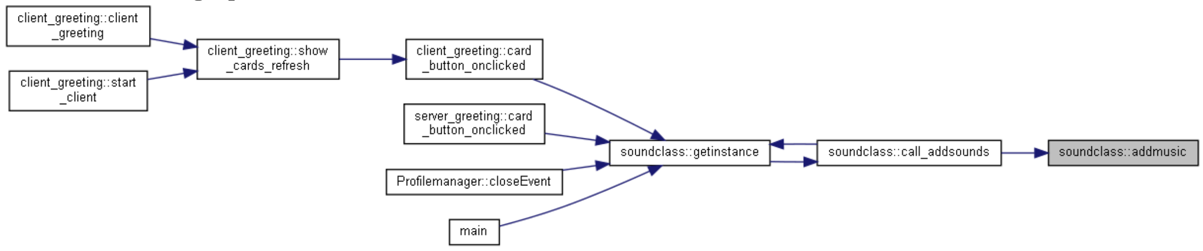
void soundclass::addeffects (const char * *path*)

Here is the caller graph for this function:


client_greeting::client_greeting
client_greeting::start_client
client_greeting::show_cards_refresh
client_greeting::card_button_onclicked
server_greeting::card_button_onclicked
Profilemanager::closeEvent
main
soundclass::getinstance
soundclass::call_addsounds
soundclass::addeffects

void soundclass::addmusic (const char * *path*)

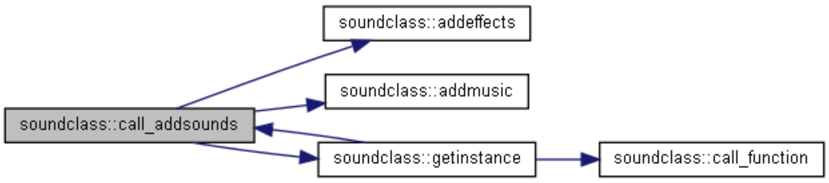
Here is the caller graph for this function:



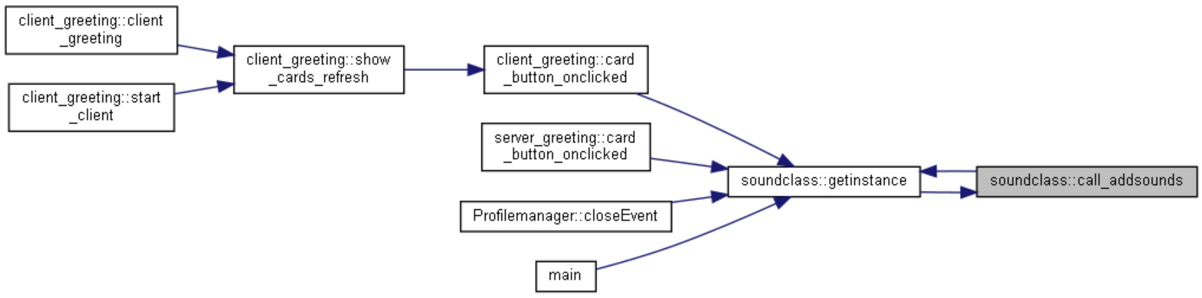
client_greeting::client_greeting
client_greeting::start_client
client_greeting::show_cards_refresh
client_greeting::card_button_onclicked
server_greeting::card_button_onclicked
Profilemanager::closeEvent
main
soundclass::getinstance
soundclass::call_addsounds
soundclass::addmusic

void soundclass::call_addsounds ()

Here is the call graph for this function:



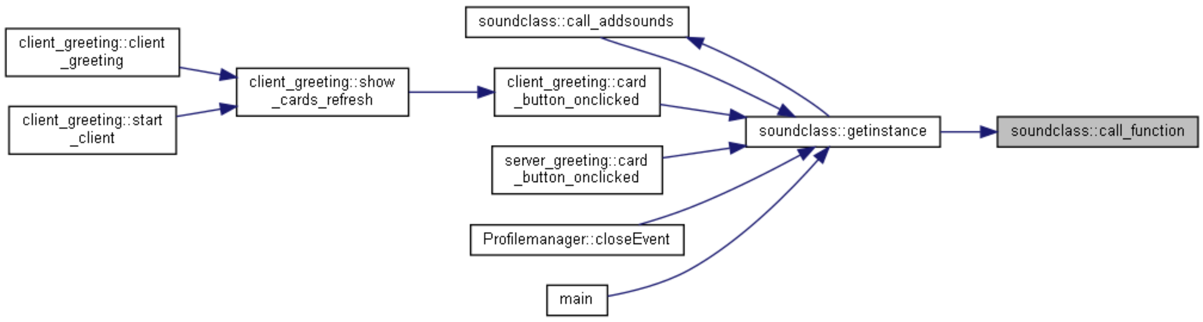
Here is the caller graph for this function:



client_greeting::client_greeting
client_greeting::start_client
client_greeting::show_cards_refresh
client_greeting::card_button_onclicked
server_greeting::card_button_onclicked
Profilemanager::closeEvent
main
soundclass::getinstance
soundclass::call_addsounds

void soundclass::call_function () [inline]

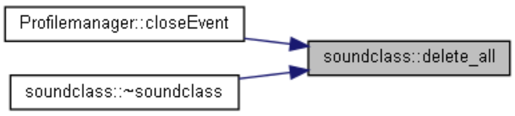
Here is the caller graph for this function:



soundclass::call_addsounds
client_greeting::client_greeting
client_greeting::start_client
client_greeting::show_cards_refresh
client_greeting::card_button_onclicked
server_greeting::card_button_onclicked
Profilemanager::closeEvent
main
soundclass::getinstance
soundclass::call_function

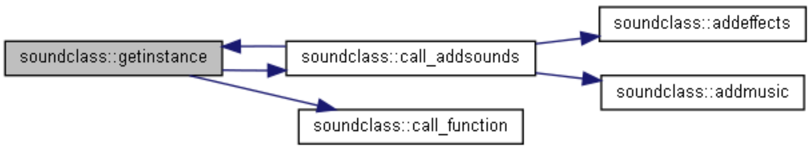
void soundclass::delete_all ()

Here is the caller graph for this function:

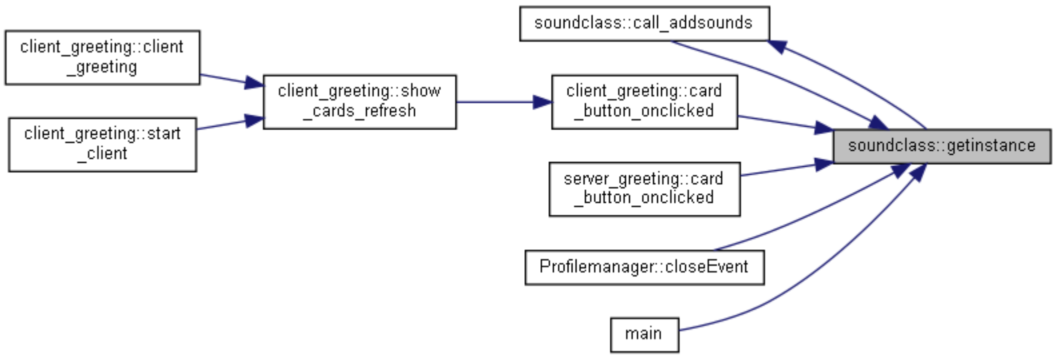


static soundclass* soundclass::getinstance () [inline], [static]

Here is the call graph for this function:

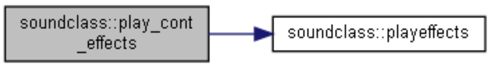


Here is the caller graph for this function:


client_greeting::client_greeting
client_greeting::start_client
client_greeting::show_cards_refresh
soundclass::call_addsounds
client_greeting::card_button_onclicked
server_greeting::card_button_onclicked
Profilemanager::closeEvent
main
soundclass::getinstance

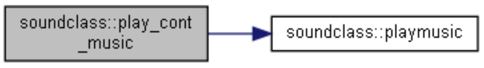
void soundclass::play_cont_effects (int *num*)

Here is the call graph for this function:



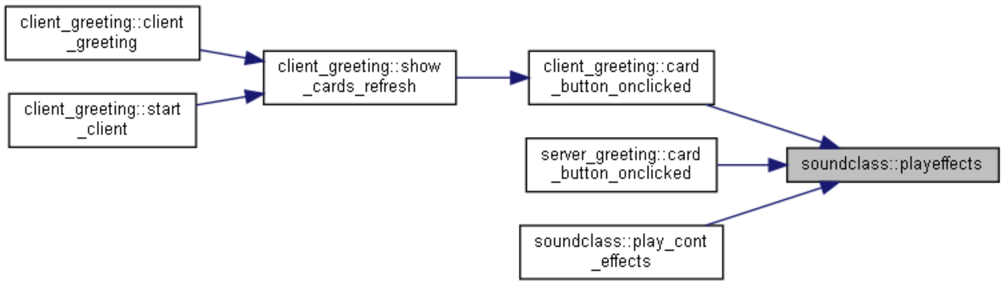
void soundclass::play_cont_music (int *num*)

Here is the call graph for this function:



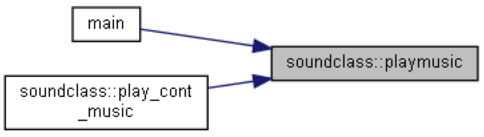
void soundclass::playeffects (int *num*) const

Here is the caller graph for this function:


client_greeting::client_greeting
client_greeting::start_client
client_greeting::show_cards_refresh
client_greeting::card_button_onclicked
server_greeting::card_button_onclicked
soundclass::play_cont_effects
soundclass::playeffects

void soundclass::playmusic (int *num*) const

Here is the caller graph for this function:



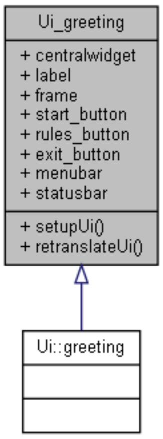
The documentation for this class was generated from the following files:

- **soundclass.h**
- **soundclass.cpp**

Ui_greeting Class Reference

`#include <greetings.h>`

Inheritance diagram for Ui_greeting:



Collaboration diagram for Ui_greeting:

Ui_greeting
+ centralwidget
+ label
+ frame
+ start_button
+ rules_button
+ exit_button
+ menubar
+ statusbar
+ setupUi()
+ retranslateUi()

Public Member Functions

- void **setupUi** (QMainWindow ***greeting**)
- void **retranslateUi** (QMainWindow ***greeting**)

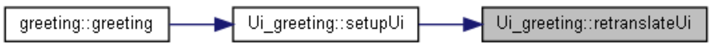
Public Attributes

- QWidget * **centralwidget**
- QLabel * **label**
- QFrame * **frame**
- QPushButton * **start_button**
- QPushButton * **rules_button**
- QPushButton * **exit_button**
- QMenuBar * **menubar**
- QStatusBar * **statusbar**

Member Function Documentation

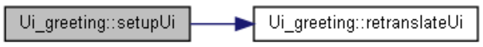
void Ui_greeting::retranslateUi (QMainWindow * *greeting*) `[inline]`

Here is the caller graph for this function:

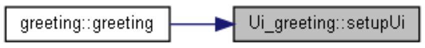


void Ui_greeting::setupUi (QMainWindow * *greeting*) [inline]

Here is the call graph for this function:



Here is the caller graph for this function:



Member Data Documentation

QWidget* Ui_greeting::centralwidget

QPushButton* Ui_greeting::exit_button

QFrame* Ui_greeting::frame

QLabel* Ui_greeting::label

QMenuBar* Ui_greeting::menubar

QPushButton* Ui_greeting::rules_button

QPushButton* Ui_greeting::start_button

QStatusBar* Ui_greeting::statusbar

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

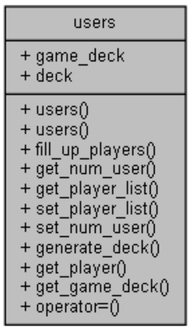
- **greetings.h**

users Class Reference

A class to generate decks of cards and store each player's decks and game_deck.

`#include <users.h>`

Collaboration diagram for users:



Public Member Functions

- **users** ()=default
- **users** (unsigned, bool)
- void **fill_up_players** (unsigned)
  This function is used to fill up the players with initial cards from the deck.
- unsigned **get_num_user** () const
  This is a simple get method to return the number of players.
- QVector< QVector< **card** > > **get_player_list** () const
  This method returns the vector of a vector of card objects representing each user's deck of cards.
- void **set_player_list** (QVector< QVector< **card** >> &)
  This method is used to set the player_list vector.
- void **set_num_user** (unsigned n)
  This method is used to set the number of players.
- QVector< **card** > **generate_deck** ()
  This method generates the entire deck of cards and invokes the fill_up_players method.
- QVector< **card** > & **get_player** (unsigned p)
  This method returns a specific player's card deck taking in the index according to player_list vector.
- QVector< **card** > & **get_game_deck** ()
- void **operator=** (const **users** &)

Public Attributes

- QVector< **card** > **game_deck**
  Variable to store the remaining generated cards after distributing to playerss.

- QVector< **card** > **deck**
  The initial container to store all the generated cards before they are split into player and game_deckcards.

Detailed Description

A class to generate decks of cards and store each player's decks and game_deck.

Constructor & Destructor Documentation

users::users () `[default]`

users::users (unsigned ***n*****, bool** ***color_drop*****)**

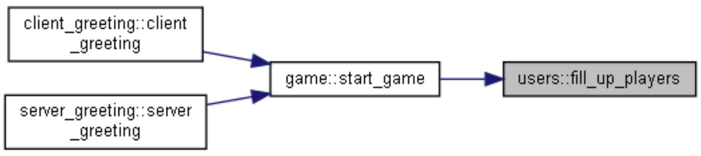
Member Function Documentation

void users::fill_up_players (unsigned ***num*****)**

This function is used to fill up the players with initial cards from the deck.

This function takes in the number of players as parameter and creates that many objects in the player_list vector and fills them with cards from a shuffled deck

Here is the caller graph for this function:


client_greeting::client_greeting
server_greeting::server_greeting
game::start_game
users::fill_up_players

QVector< card > users::generate_deck ()

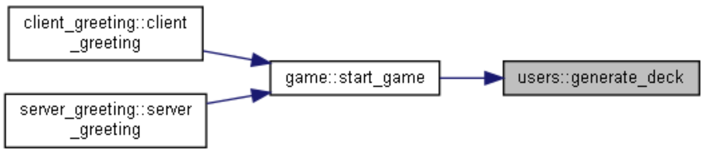
This method generates the entire deck of cards and invokes the fill_up_players method.

The deck is generated in a pre-defined fashion and is shuffled using a time based seed. Then the player

Parameters

n	denotes the number of players to be set to number_players. decks are filled up linearly from the shuffled deck

Here is the caller graph for this function:


client_greeting::client_greeting
server_greeting::server_greeting
game::start_game
users::generate_deck

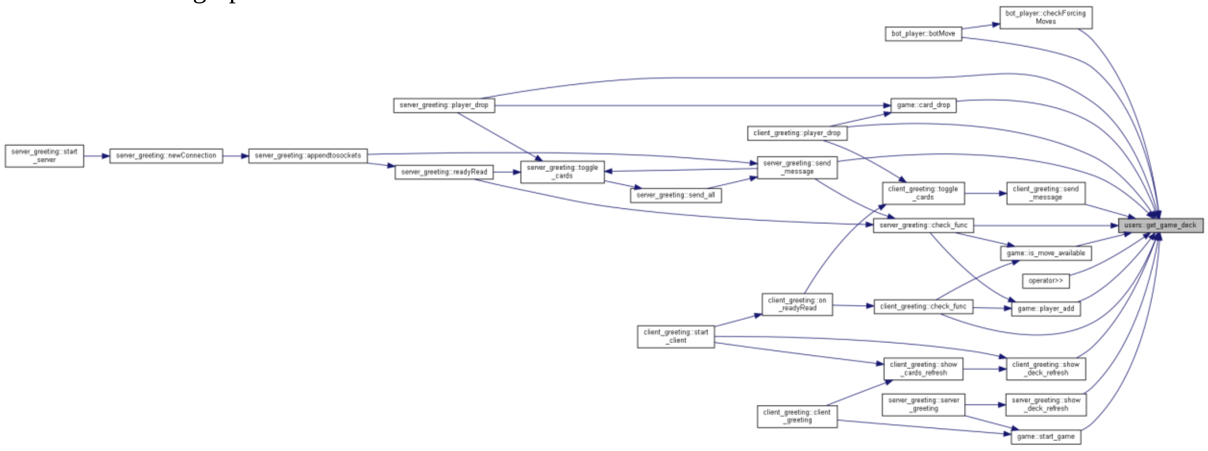
QVector< card > & users::get_game_deck ()

This method returns the game_deck which is basically the remaining cards left in the generated pool of cards after distributing to players

Parameters

p	denotes the player number(starts from 1 being server) to be picked from the player's list. Note that it is player number and not player index.

Here is the caller graph for this function:

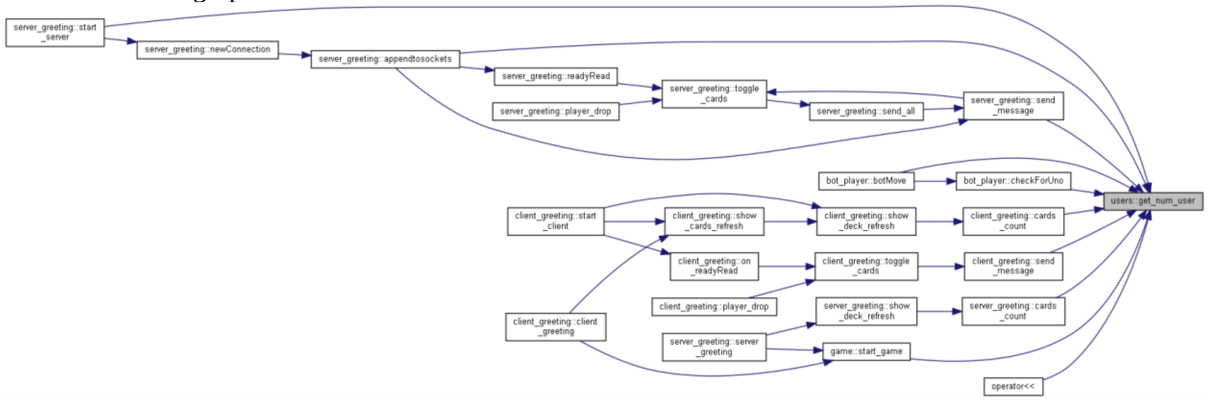


bot_player::checkForcingMoves
bot_player::botMove
server_greeting::player_drop
game::card_drop
client_greeting::player_drop
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead
server_greeting::toggle_cards
server_greeting::send_message
server_greeting::send_all
client_greeting::toggle_cards
client_greeting::send_message
server_greeting::check_func
users::get_game_deck
game::is_move_available
operator>>
client_greeting::on_readyRead
client_greeting::check_func
game::player_add
client_greeting::start_client
client_greeting::show_cards_refresh
client_greeting::show_deck_refresh
server_greeting::server_greeting
server_greeting::show_deck_refresh
client_greeting::client_greeting
game::start_game

unsigned users::get_num_user () const

This is a simple get method to return the number of players.

Here is the caller graph for this function:

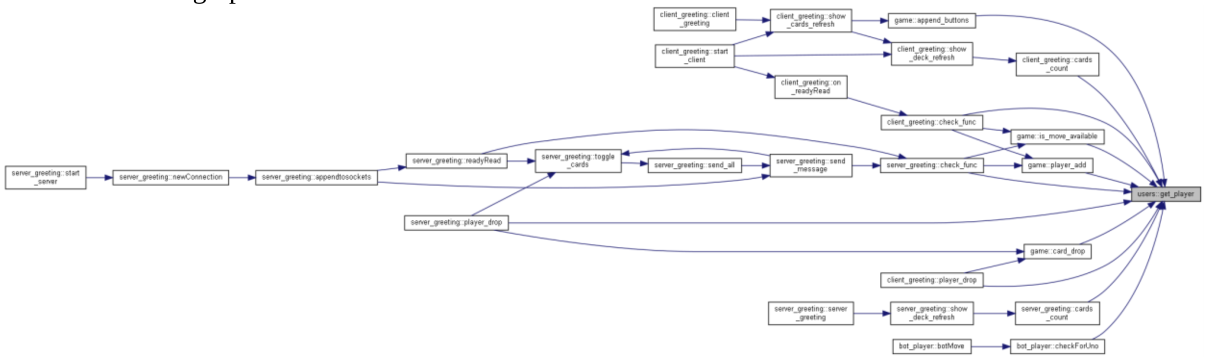


server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead
server_greeting::toggle_cards
server_greeting::player_drop
server_greeting::send_all
server_greeting::send_message
bot_player::botMove
bot_player::checkForUno
users::get_num_user
client_greeting::start_client
client_greeting::show_cards_refresh
client_greeting::show_deck_refresh
client_greeting::cards_count
client_greeting::on_readyRead
client_greeting::toggle_cards
client_greeting::send_message
client_greeting::player_drop
server_greeting::show_deck_refresh
server_greeting::cards_count
client_greeting::client_greeting
server_greeting::server_greeting
game::start_game
operator<<

QVector< card > & users::get_player (unsigned *p*)

This method returns a specific player's card deck taking in the index according to player_list vector.

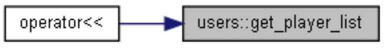
Here is the caller graph for this function:



client_greeting::client_greeting
client_greeting::show_cards_refresh
game::append_buttons
client_greeting::start_client
client_greeting::show_deck_refresh
client_greeting::cards_count
client_greeting::on_readyRead
client_greeting::check_func
game::is_move_available
server_greeting::start_server
server_greeting::newConnection
server_greeting::appendtosockets
server_greeting::readyRead
server_greeting::toggle_cards
server_greeting::send_all
server_greeting::send_message
server_greeting::check_func
game::player_add
users::get_player
server_greeting::player_drop
game::card_drop
client_greeting::player_drop
server_greeting::server_greeting
server_greeting::show_deck_refresh
server_greeting::cards_count
bot_player::botMove
bot_player::checkForUno

QVector< QVector< card > > users::get_player_list () const

This method returns the vector of a vector of card objects representing each user's deck of cards.
Here is the caller graph for this function:



void users::operator= (const users & *u*)

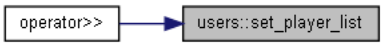
void users::set_num_user (unsigned *n*)

This method is used to set the number of players.
Here is the caller graph for this function:

operator>>
users::set_num_user

void users::set_player_list (QVector< QVector< card >> & *list*)

This method is used to set the player_list vector.
Here is the caller graph for this function:



Member Data Documentation

QVector<card> users::deck

The initial container to store all the generated cards before they are split into player and game_deck cards.

QVector<card> users::game_deck

Variable to store the remaining generated cards after distributing to playerss.

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

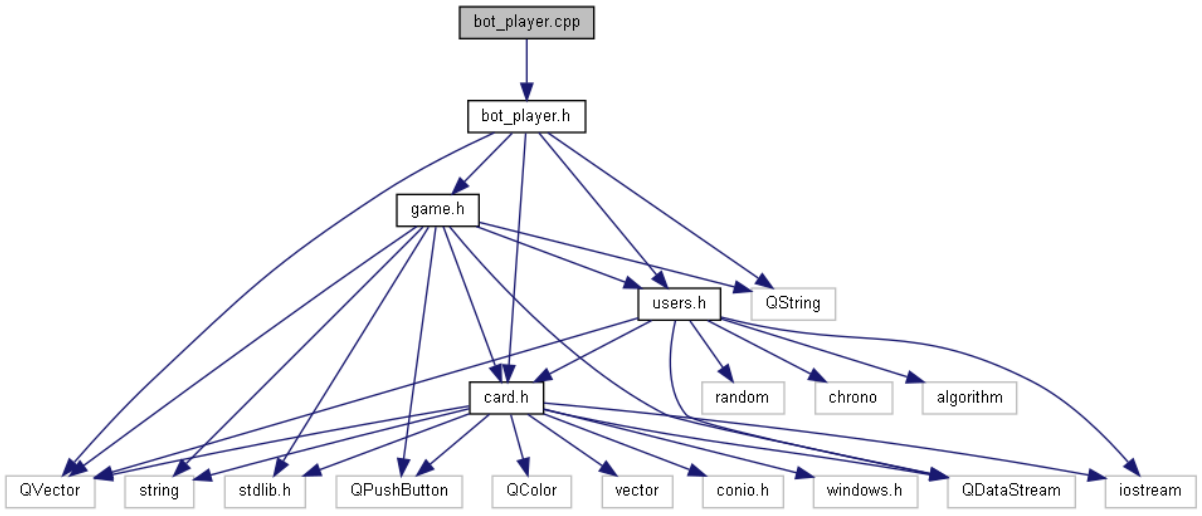
- **users.h**
- **users.cpp**

File Documentation

bot_player.cpp File Reference

`#include "bot_player.h"`

Include dependency graph for bot_player.cpp:



Macros

- #define **HAND_THRESHOLD** 5

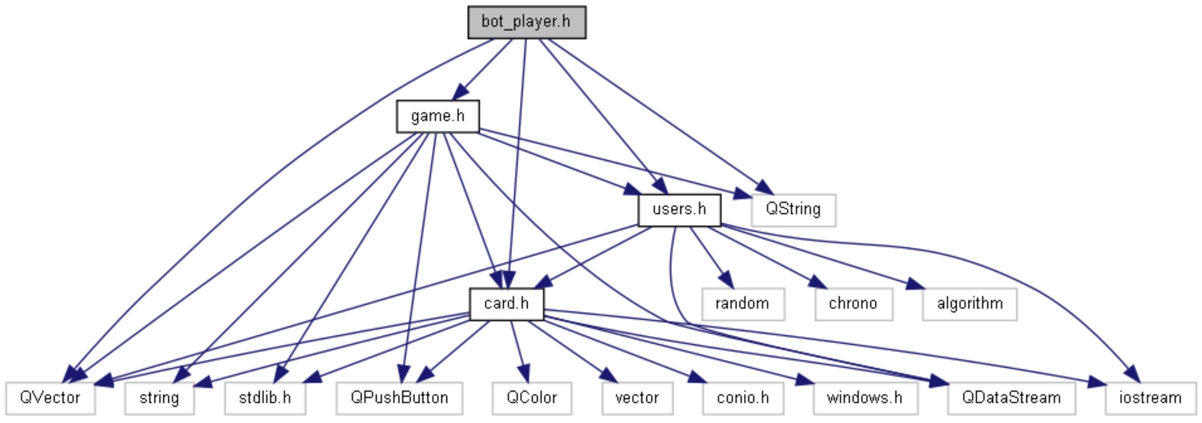
Macro Definition Documentation

#define HAND_THRESHOLD 5

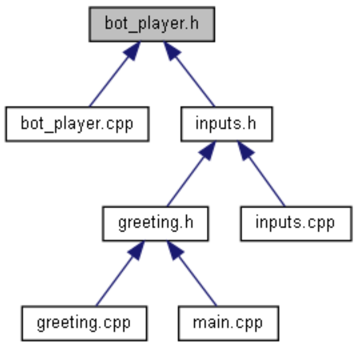
bot_player.h File Reference

```
#include "card.h"
#include "game.h"
#include "users.h"
#include <QVector>
#include <QString>
```

Include dependency graph for bot_player.h:



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



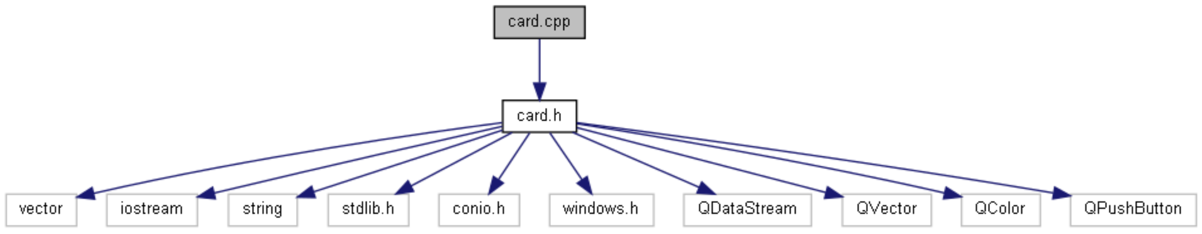
Classes

- class **bot_player**

card.cpp File Reference

`#include "card.h"`

Include dependency graph for card.cpp:



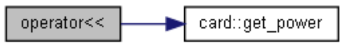
Functions

- QDataStream & **operator**<< (QDataStream &out, const **card** &c)
- QDataStream & **operator**>> (QDataStream &in, **card** &c)

Function Documentation

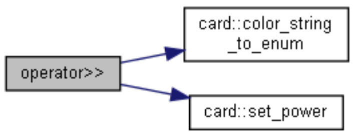
QDataStream& operator<< (QDataStream & *out*, const card & *c*)

Here is the call graph for this function:



QDataStream& operator>> (QDataStream & *in*, card & *c*)

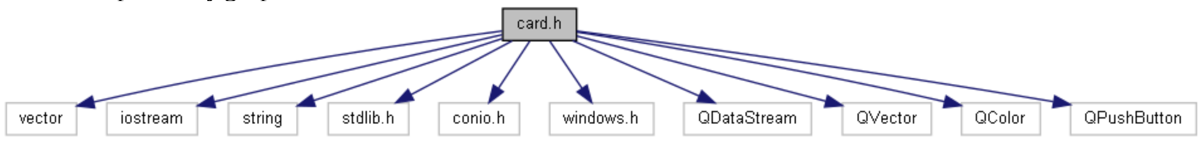
Here is the call graph for this function:



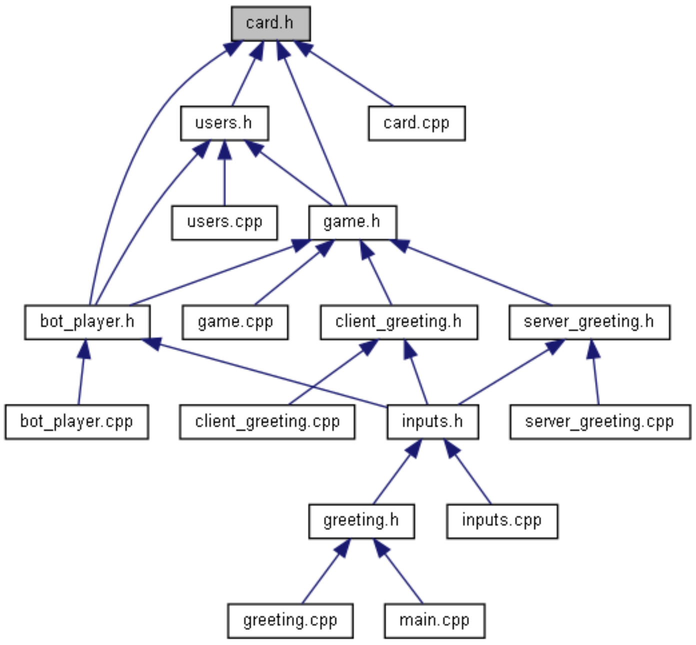
card.h File Reference

```
#include <vector>
#include <iostream>
#include <string>
#include <stdlib.h>
#include <conio.h>
#include <windows.h>
#include <QDataStream>
#include <QVector>
#include <QColor>
#include <QPushButton>
```

Include dependency graph for card.h:



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



Classes

- class **card**

Enumerations

- enum class **color_t** { **RED**, **BLUE**, **GREEN**, **YELLOW**, **NO_COLOR** }
  Enum class to represent color of a UNO card.
- enum class **number_t** { **ZERO**, **ONE**, **TWO**, **THREE**, **FOUR**, **FIVE**, **SIX**, **SEVEN**, **EIGHT**, **NINE**, **TEN**, **SKIP**, **REVERSE**, **DRAW_TWO**, **WILD**, **DRAW_FOUR_WILD**, **COLOR_DROP**, **NO_NUMBER** }
  Enum class to represent the number present in the UNO card.

Functions

- QDataStream & **operator**<< (QDataStream &out, const **card** &c)
- QDataStream & **operator**>> (QDataStream &in, **card** &c)

Enumeration Type Documentation

enum color_t `[strong]`

Enum class to represent color of a UNO card.

Enumerator:

RED	
BLUE	
GREEN	
YELLOW	
NO_COLOR	

enum number_t `[strong]`

Enum class to represent the number present in the UNO card.

After the number ten the enum class has numbers for special cards such as SKIP, REVERSE etc.

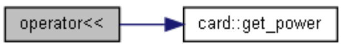
Enumerator:

ZERO	
ONE	
TWO	
THREE	
FOUR	
FIVE	
SIX	
SEVEN	
EIGHT	
NINE	
TEN	
SKIP	
REVERSE	
DRAW_TWO	
WILD	
DRAW_FOUR_WILD	
COLOR_DROP	
NO_NUMBER	

Function Documentation

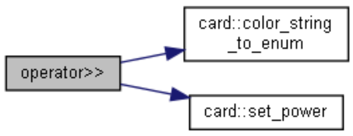
QDataStream& operator<< (QDataStream & *out*, const card & *c*)

Here is the call graph for this function:



QDataStream& operator>> (QDataStream & *in*, card & *c*)

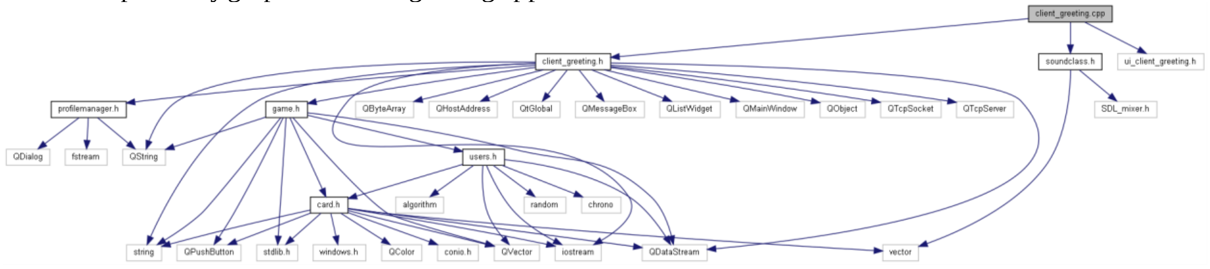
Here is the call graph for this function:



client_greeting.cpp File Reference

```
#include "client_greeting.h"
#include "ui_client_greeting.h"
#include "soundclass.h"
```

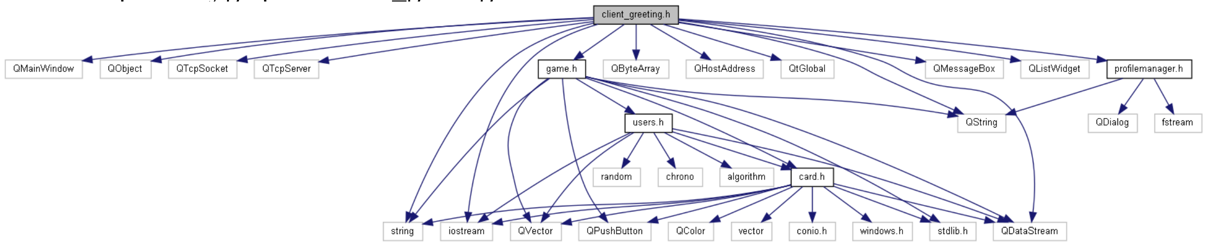
Include dependency graph for client_greeting.cpp:



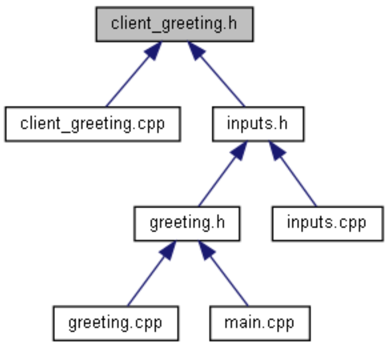
client_greeting.h File Reference

```
#include <QMainWindow>
#include <QObject>
#include <QTcpSocket>
#include <QTcpServer>
#include <iostream>
#include <QString>
#include <string>
#include "QByteArray"
#include <QHostAddress>
#include "QtGlobal"
#include "game.h"
#include <QDataStream>
#include <QMessageBox>
#include <QListWidget>
#include "profilemanager.h"
```

Include dependency graph for client_greeting.h:



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



Classes

- class **client_greeting**
  A class to handle the gameplay for client players.

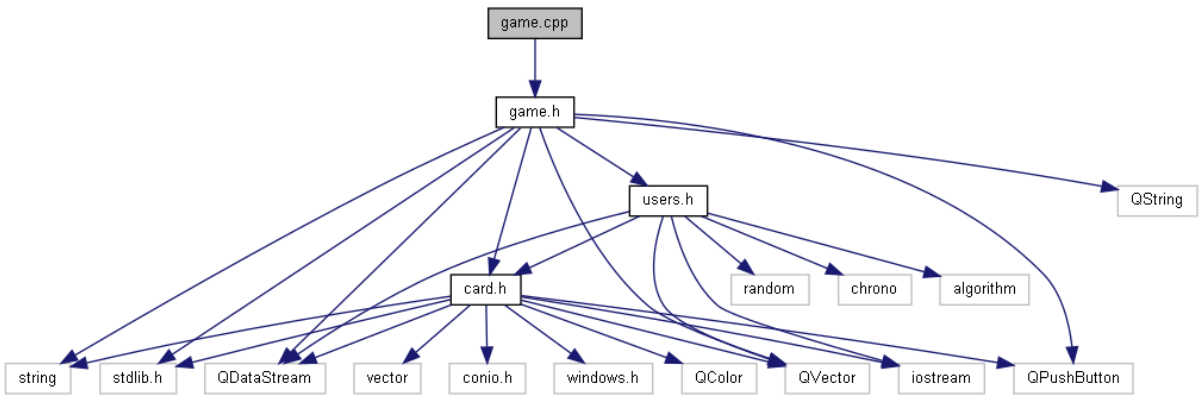
Namespaces

- **Ui**

game.cpp File Reference

`#include "game.h"`

Include dependency graph for game.cpp:



Functions

- QDataStream & **operator**<< (QDataStream &out, const **game** &g)
- QDataStream & **operator**>> (QDataStream &in, **game** &g)

Function Documentation

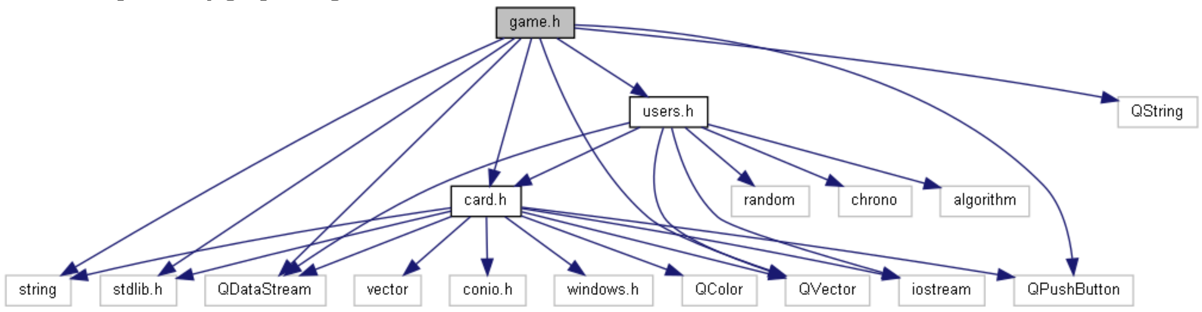
QDataStream& operator<< (QDataStream & *out*, const game & *g*)

QDataStream& operator>> (QDataStream & *in*, game & *g*)

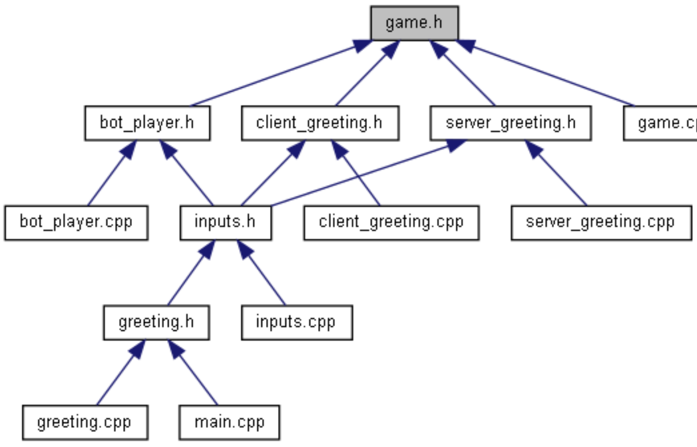
game.h File Reference

```
#include "users.h"
#include "card.h"
#include <QVector>
#include <string>
#include <stdlib.h>
#include <QDataStream>
#include <QString>
#include <QPushButton>
```

Include dependency graph for game.h:



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



Classes

- class **game**
  A Class to handle game variables common to both client and server.

Functions

- QDataStream & **operator<<** (QDataStream &out, const **game** &g)
- QDataStream & **operator>>** (QDataStream &in, **game** &g)

Function Documentation

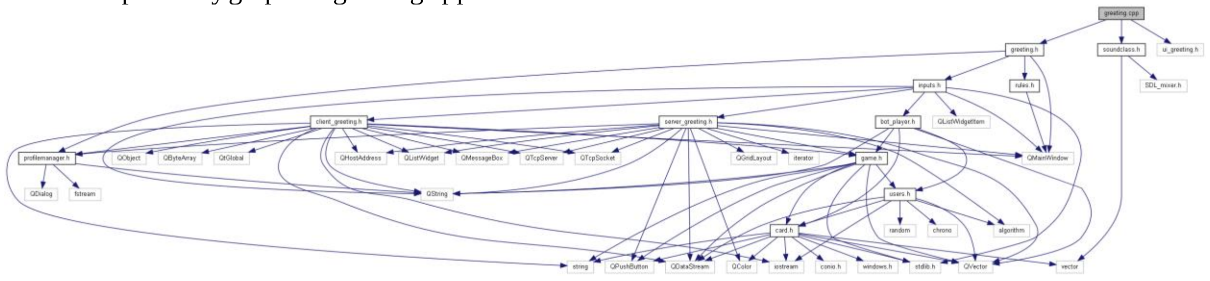
QDataStream& operator<< (QDataStream & *out*, const game & *g*)

QDataStream& operator>> (QDataStream & *in*, game & *g*)

greeting.cpp File Reference

```
#include "greeting.h"
#include "ui_greeting.h"
#include "soundclass.h"
```

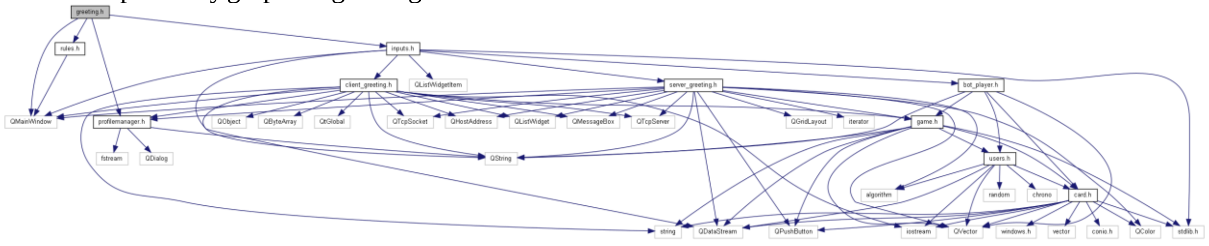
Include dependency graph for greeting.cpp:



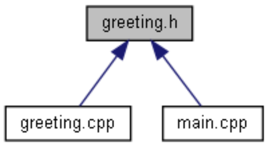
greeting.h File Reference

```
#include <QMainWindow>
#include "rules.h"
#include "inputs.h"
#include "profilemanager.h"
```

Include dependency graph for greeting.h:



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



Classes

- class **greeting**
  A GUI class to display the user with navigations for start, rules, profile and exit options.

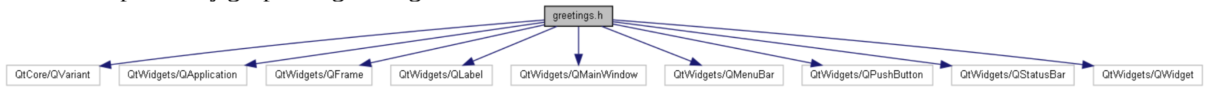
Namespaces

- **Ui**

greetings.h File Reference

```
#include <QtCore/QVariant>
#include <QtWidgets/QApplication>
#include <QtWidgets/QFrame>
#include <QtWidgets/QLabel>
#include <QtWidgets/QMainWindow>
#include <QtWidgets/QMenuBar>
#include <QtWidgets/QPushButton>
#include <QtWidgets/QStatusBar>
#include <QtWidgets/QWidget>
```

Include dependency graph for greetings.h:



Classes

- class **Ui_greeting**
- class **Ui::greeting**

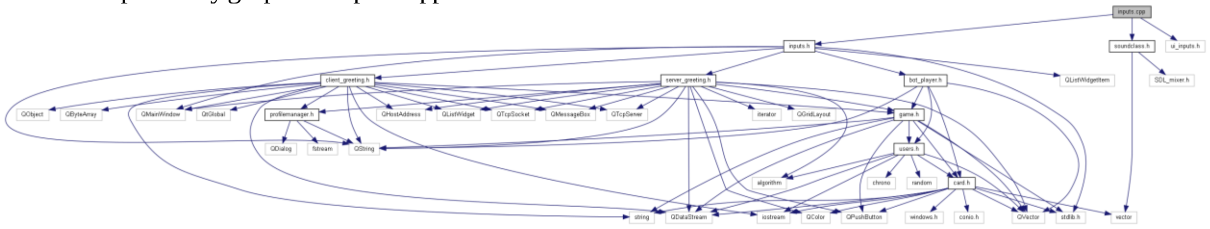
Namespaces

- **Ui**

inputs.cpp File Reference

```
#include "inputs.h"
#include "ui_inputs.h"
#include "soundclass.h"
```

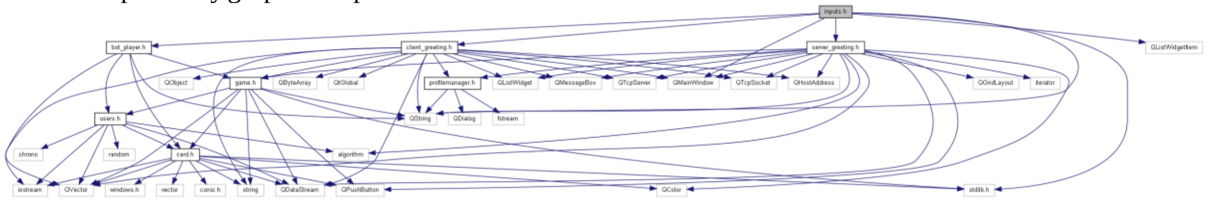
Include dependency graph for inputs.cpp:



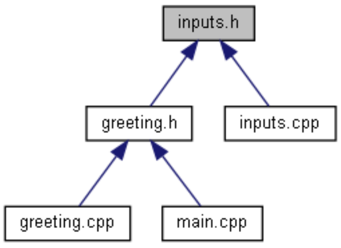
inputs.h File Reference

```
#include <QMainWindow>
#include "server_greeting.h"
#include "client_greeting.h"
#include "bot_player.h"
#include <QListWidgetItem>
#include <stdlib.h>
#include <QString>
```

Include dependency graph for inputs.h:



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



Classes

- class **inputs**
  A GUI class to let the user enter the game settings like number of players and player_type.

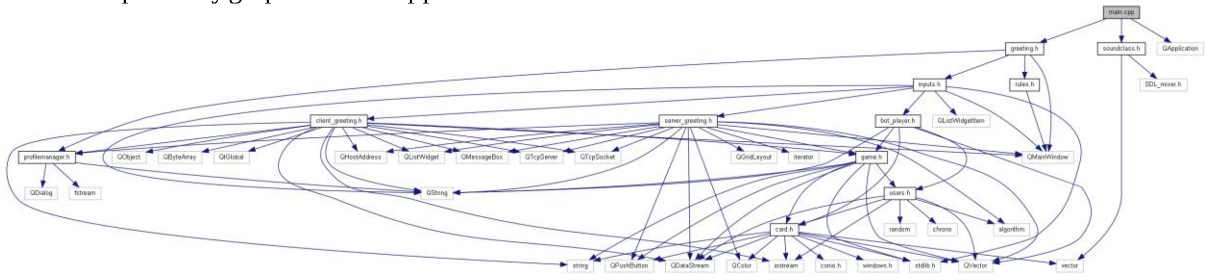
Namespaces

- **Ui**

main.cpp File Reference

```
#include "greeting.h"
#include <QApplication>
#include "soundclass.h"
```

Include dependency graph for main.cpp:



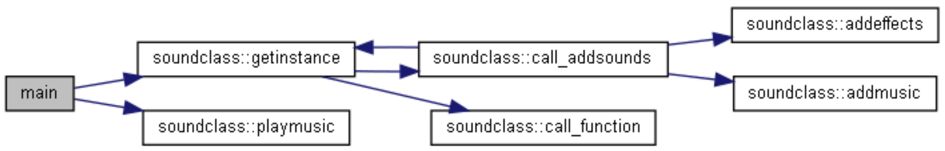
Functions

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

Function Documentation

int main (int *argc*, char * *argv*[])

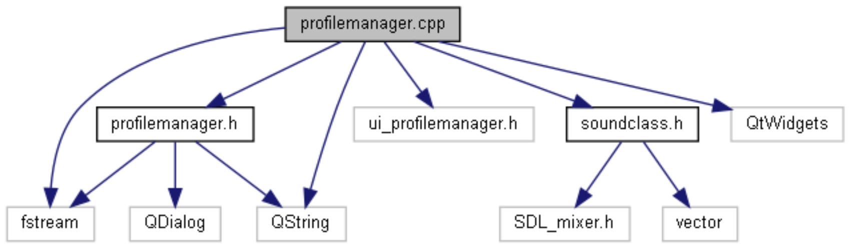
Here is the call graph for this function:



profilemanager.cpp File Reference

```
#include "profilemanager.h"
#include "ui_profilemanager.h"
#include <fstream>
#include <QString>
#include "soundclass.h"
#include <QtWidgets>
```

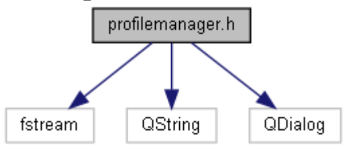
Include dependency graph for profilemanager.cpp:



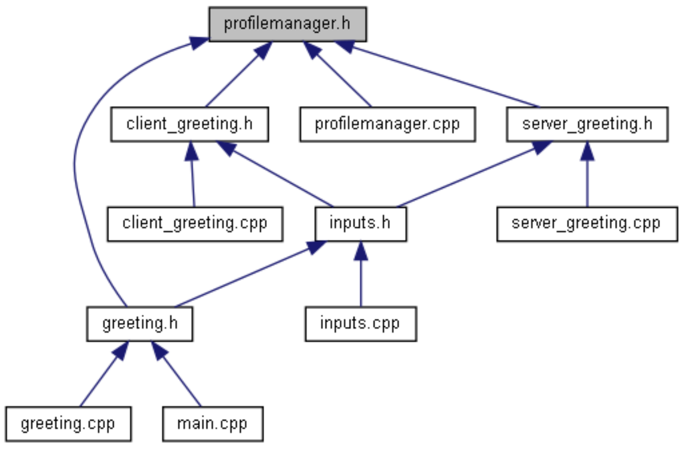
profilemanager.h File Reference

```
#include <fstream>
#include <QString>
#include <QDialog>
```

Include dependency graph for profilemanager.h:



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



Classes

- class **Profilemanager**

Namespaces

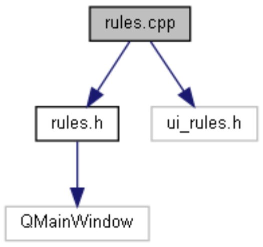
- **Ui**

README.md File Reference

rules.cpp File Reference

```
#include "rules.h"
#include "ui_rules.h"
```

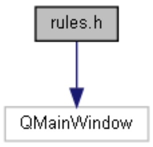
Include dependency graph for rules.cpp:



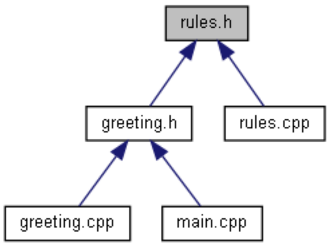
rules.h File Reference

`#include <QMainWindow>`

Include dependency graph for rules.h:



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



Classes

- class **rules**
  A class to display the rules of the game for the user.

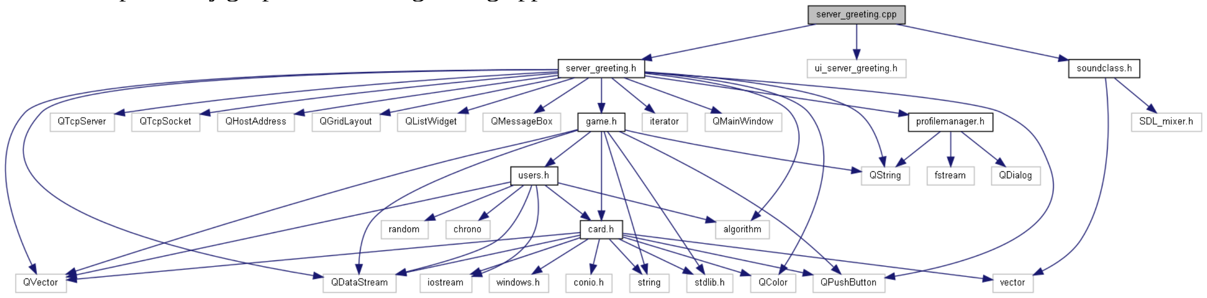
Namespaces

- **Ui**

server_greeting.cpp File Reference

```
#include "server_greeting.h"
#include "ui_server_greeting.h"
#include "soundclass.h"
```

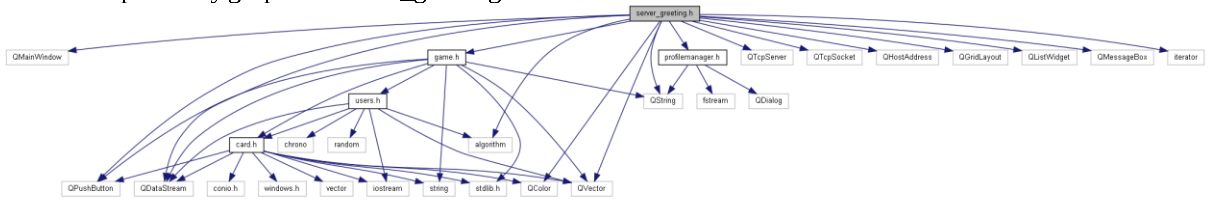
Include dependency graph for server_greeting.cpp:



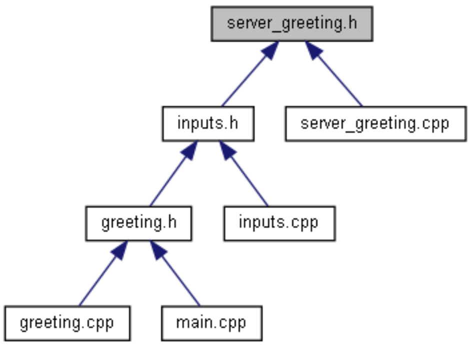
server_greeting.h File Reference

```
#include <QMainWindow>
#include "game.h"
#include <QVector>
#include <QTcpServer>
#include <QTcpSocket>
#include <QDataStream>
#include "QHostAddress"
#include <QString>
#include <QPushButton>
#include <QGridLayout>
#include <QColor>
#include <QListWidget>
#include <QMessageBox>
#include "profilemanager.h"
#include <algorithm>
#include <iterator>
```

Include dependency graph for server_greeting.h:



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



Classes

- class **server_greeting**
  A class to handle the gameplay for the server player.

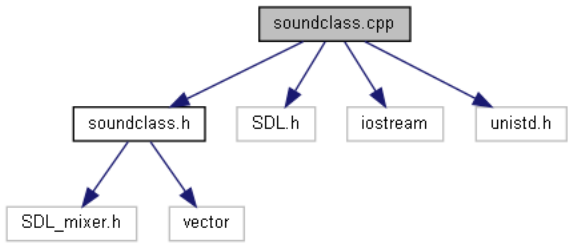
Namespaces

- **Ui**

soundclass.cpp File Reference

```
#include "soundclass.h"
#include <SDL.h>
#include <iostream>
#include <unistd.h>
```

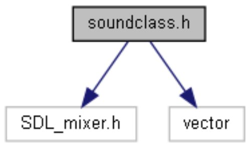
Include dependency graph for soundclass.cpp:



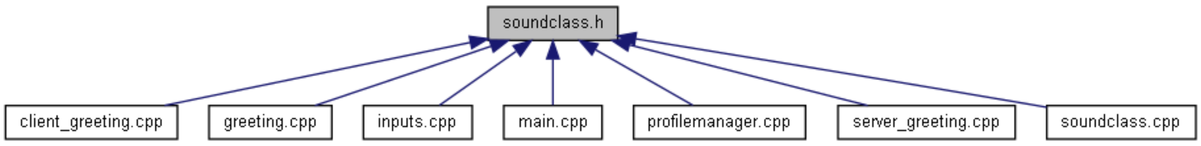
soundclass.h File Reference

```
#include <SDL_mixer.h>
#include <vector>
```

Include dependency graph for soundclass.h:



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



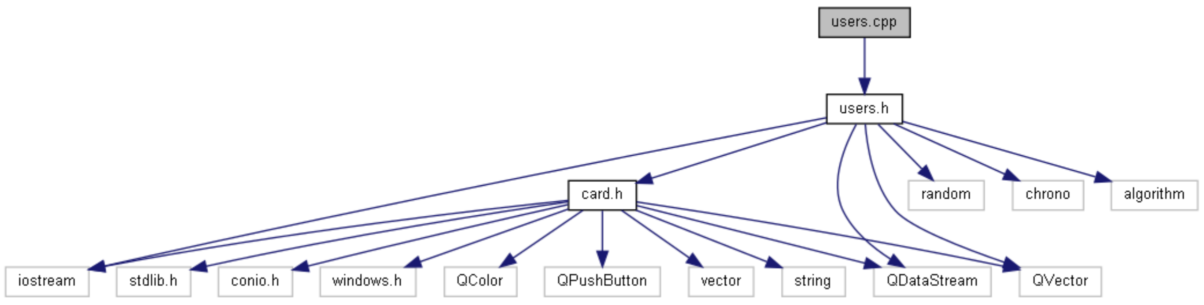
Classes

- class **soundclass**

users.cpp File Reference

`#include "users.h"`

Include dependency graph for users.cpp:



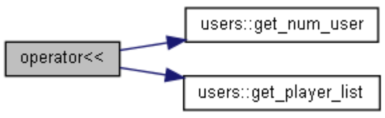
Functions

- QDataStream & **operator<<** (QDataStream &out, const **users** &u)
- QDataStream & **operator>>** (QDataStream &in, **users** &u)

Function Documentation

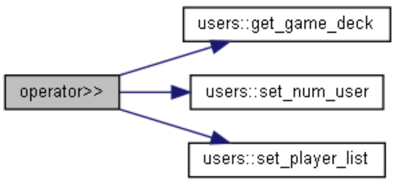
QDataStream& operator<< (QDataStream & *out*, const users & *u*)

Here is the call graph for this function:



QDataStream& operator>> (QDataStream & *in*, users & *u*)

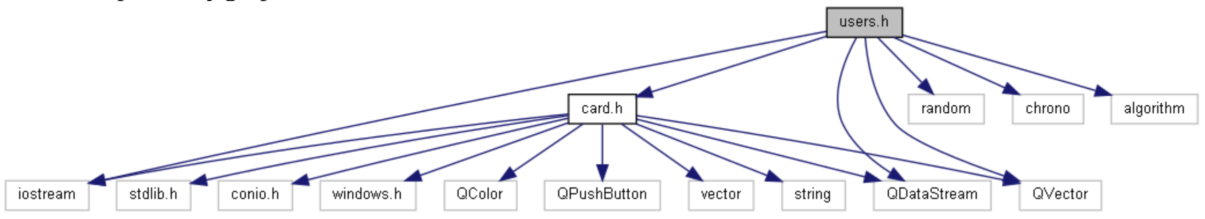
Here is the call graph for this function:



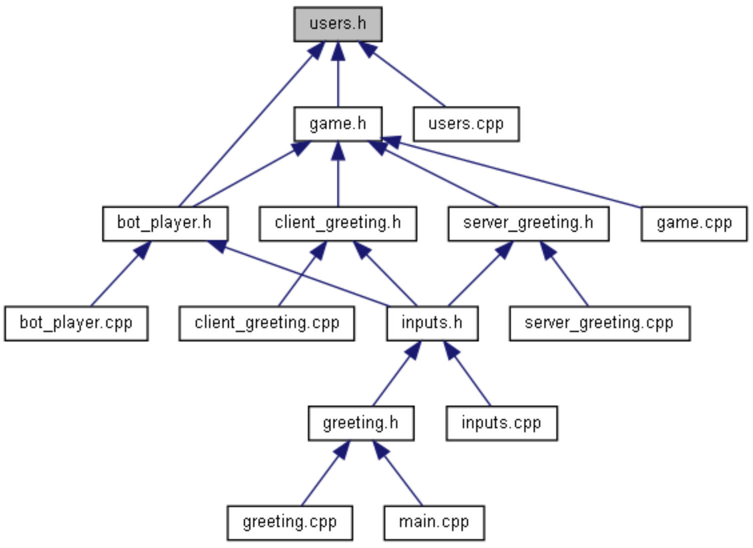
users.h File Reference

```
#include "card.h"
#include <random>
#include <chrono>
#include <algorithm>
#include <iostream>
#include <QDataStream>
#include <QVector>
```

Include dependency graph for users.h:



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



Classes

- class **users**
  A class to generate decks of cards and store each player's decks and game_deck.

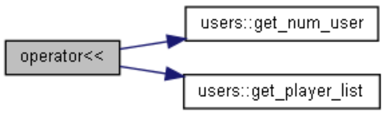
Functions

- QDataStream & **operator<<** (QDataStream &out, const **users** &u)
- QDataStream & **operator>>** (QDataStream &in, **users** &u)

Function Documentation

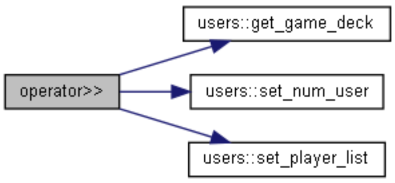
QDataStream& operator<< (QDataStream & *out*, const users & *u*)

Here is the call graph for this function:



QDataStream& operator>> (QDataStream & *in*, users & *u*)

Here is the call graph for this function:



Index

INDEX