

## C++Names

Generated by Doxygen 1.13.2



<b>1 Hierarchical Index</b>	<b>1</b>
1.1 Class Hierarchy	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 GameBoard::Card Struct Reference	7
4.1.1 Detailed Description	7
4.1.2 Member Data Documentation	7
4.1.2.1 revealed	7
4.1.2.2 type	7
4.1.2.3 word	7
4.2 ChatBox Class Reference	8
4.2.1 Detailed Description	9
4.2.2 Member Enumeration Documentation	9
4.2.2.1 Team	9
4.2.3 Constructor & Destructor Documentation	10
4.2.3.1 ChatBox()	10
4.2.3.2 ~ChatBox()	10
4.2.4 Member Function Documentation	10
4.2.4.1 addPlayerMessage()	10
4.2.4.2 addSystemMessage()	11
4.2.4.3 clearChat()	11
4.2.4.4 limitReachedMessage()	11
4.2.4.5 massSend	11
4.2.4.6 sendMessage	12
4.2.4.7 setPlayerName()	12
4.2.5 Member Data Documentation	12
4.2.5.1 chatDisplay	12
4.2.5.2 chatInput	12
4.2.5.3 playerName	12
4.2.5.4 sendButton	13
4.2.5.5 team	13
4.3 CreateAccountWindow Class Reference	13
4.3.1 Detailed Description	15
4.3.2 Constructor & Destructor Documentation	15
4.3.2.1 CreateAccountWindow()	15
4.3.3 Member Function Documentation	15
4.3.3.1 accountCreated	15

4.3.3.2 back	15
4.3.3.3 getInstance()	15
4.3.3.4 goBack	16
4.3.3.5 onCreateAccountClicked	16
4.3.3.6 saveJsonFile()	16
4.3.3.7 setPreviousScreen()	16
4.3.3.8 show	16
4.3.4 Member Data Documentation	17
4.3.4.1 createAccountButton	17
4.3.4.2 instance	17
4.3.4.3 jsonFilePath	17
4.3.4.4 previousScreen	17
4.3.4.5 statusLabel	17
4.3.4.6 usernameEdit	17
4.4 GameBoard Class Reference	18
4.4.1 Detailed Description	21
4.4.2 Member Enumeration Documentation	21
4.4.2.1 CardType	21
4.4.2.2 Turn	21
4.4.3 Constructor & Destructor Documentation	22
4.4.3.1 GameBoard()	22
4.4.3.2 ~GameBoard()	22
4.4.4 Member Function Documentation	23
4.4.4.1 checkGameEnd()	23
4.4.4.2 displayGuess	23
4.4.4.3 displayHint	23
4.4.4.4 endGame()	23
4.4.4.5 gameEnded	24
4.4.4.6 generateGameGrid()	24
4.4.4.7 loadWordsFromFile()	24
4.4.4.8 nextTurn()	25
4.4.4.9 onCardClicked()	25
4.4.4.10 onContinueClicked()	25
4.4.4.11 resetGame()	25
4.4.4.12 setBlueOperativeName()	25
4.4.4.13 setBlueSpyMasterName()	26
4.4.4.14 setRedOperativeName()	26
4.4.4.15 setRedSpyMasterName()	26
4.4.4.16 setupUI()	27
4.4.4.17 show	27
4.4.4.18 showTransition()	27
4.4.4.19 updateScores()	28

4.4.4.20 updateTeamLabels()	28
4.4.5 Member Data Documentation	28
4.4.5.1 blueCardsRemaining	28
4.4.5.2 blueOperativeName	28
4.4.5.3 blueScoreLabel	28
4.4.5.4 blueSpyMasterName	28
4.4.5.5 blueTeamLabel	29
4.4.5.6 cards	29
4.4.5.7 chatBox	29
4.4.5.8 correspondingNumber	29
4.4.5.9 currentGuesses	29
4.4.5.10 currentHint	29
4.4.5.11 currentPlayerName	29
4.4.5.12 currentPlayerTeam	29
4.4.5.13 currentTurn	30
4.4.5.14 currentTurnLabel	30
4.4.5.15 gameGrid	30
4.4.5.16 GRID_SIZE	30
4.4.5.17 gridLayout	30
4.4.5.18 maxGuesses	30
4.4.5.19 operatorGuess	30
4.4.5.20 redCardsRemaining	30
4.4.5.21 redOperativeName	31
4.4.5.22 redScoreLabel	31
4.4.5.23 redSpyMasterName	31
4.4.5.24 redTeamLabel	31
4.4.5.25 spymasterHint	31
4.4.5.26 transition	31
4.4.5.27 users	31
4.4.5.28 wordList	32
4.5 MainWindow Class Reference	32
4.5.1 Detailed Description	34
4.5.2 Constructor & Destructor Documentation	34
4.5.2.1 MainWindow()	34
4.5.2.2 ~MainWindow()	34
4.5.3 Member Function Documentation	34
4.5.3.1 openCreateAccount	34
4.5.3.2 openMultiMain	34
4.5.3.3 openPreGame	34
4.5.3.4 openStatsWindow	35
4.5.3.5 openTutorial	35
4.5.3.6 showMainWindow	35

4.5.4 Member Data Documentation	35
4.5.4.1 centralWidget	35
4.5.4.2 createAccountButton	35
4.5.4.3 createAccountWindow	35
4.5.4.4 layout	35
4.5.4.5 localPlayButton	36
4.5.4.6 onlinePlayButton	36
4.5.4.7 preGameWindow	36
4.5.4.8 statsButton	36
4.5.4.9 statsWindow	36
4.5.4.10 titleLabel	36
4.5.4.11 tutorialButton	36
4.5.4.12 tutorialWindow	37
4.6 OperatorGuess Class Reference	37
4.6.1 Detailed Description	38
4.6.2 Constructor & Destructor Documentation	38
4.6.2.1 OperatorGuess()	38
4.6.2.2 ~OperatorGuess()	39
4.6.3 Member Function Documentation	39
4.6.3.1 guessSubmitted	39
4.6.3.2 reset()	39
4.6.3.3 submitGuess	40
4.6.4 Member Data Documentation	40
4.6.4.1 submitGuessButton	40
4.7 PreGame Class Reference	40
4.7.1 Detailed Description	43
4.7.2 Constructor & Destructor Documentation	43
4.7.2.1 PreGame()	43
4.7.2.2 ~PreGame()	43
4.7.3 Member Function Documentation	43
4.7.3.1 backToMainWindow	43
4.7.3.2 getBlueTeamOperativeNickname()	43
4.7.3.3 getBlueTeamSpyMasterNickname()	44
4.7.3.4 getRedTeamOperativeNickname()	44
4.7.3.5 getRedTeamSpyMasterNickname()	44
4.7.3.6 goBackToMain	44
4.7.3.7 handleGameEnd	44
4.7.3.8 openCreateAccount	44
4.7.3.9 populateUserDropdowns()	45
4.7.3.10 show	45
4.7.3.11 start	45
4.7.3.12 startGame	45

4.7.3.13 update . . . . .	45
4.7.4 Member Data Documentation . . . . .	45
4.7.4.1 backButton . . . . .	45
4.7.4.2 blueTeamLayout . . . . .	45
4.7.4.3 blueTeamOperativeComboBox . . . . .	46
4.7.4.4 blueTeamSpyMasterComboBox . . . . .	46
4.7.4.5 buttonsLayout . . . . .	46
4.7.4.6 createAccountButton . . . . .	46
4.7.4.7 createAccountWindow . . . . .	46
4.7.4.8 gameBoard . . . . .	46
4.7.4.9 label . . . . .	46
4.7.4.10 layout . . . . .	46
4.7.4.11 redTeamLayout . . . . .	47
4.7.4.12 redTeamOperativeComboBox . . . . .	47
4.7.4.13 redTeamSpyMasterComboBox . . . . .	47
4.7.4.14 startButton . . . . .	47
4.7.4.15 teamsLayout . . . . .	47
4.7.4.16 usernames . . . . .	47
4.7.4.17 users . . . . .	47
4.8 SpymasterHint Class Reference . . . . .	48
4.8.1 Detailed Description . . . . .	49
4.8.2 Constructor & Destructor Documentation . . . . .	49
4.8.2.1 SpymasterHint() . . . . .	49
4.8.2.2 ~SpymasterHint() . . . . .	50
4.8.3 Member Function Documentation . . . . .	50
4.8.3.1 hintSubmitted . . . . .	50
4.8.3.2 reset() . . . . .	50
4.8.3.3 submitHint . . . . .	50
4.8.3.4 textToUppercase . . . . .	50
4.8.3.5 updateButtonClickable . . . . .	51
4.8.4 Member Data Documentation . . . . .	51
4.8.4.1 giveClueButton . . . . .	51
4.8.4.2 hintLineEdit . . . . .	51
4.8.4.3 numberSpinBox . . . . .	51
4.8.4.4 textValidator . . . . .	51
4.9 StatisticsWindow Class Reference . . . . .	52
4.9.1 Detailed Description . . . . .	54
4.9.2 Constructor & Destructor Documentation . . . . .	54
4.9.2.1 StatisticsWindow() . . . . .	54
4.9.2.2 ~StatisticsWindow() . . . . .	54
4.9.3 Member Function Documentation . . . . .	54
4.9.3.1 backToMainWindow . . . . .	54

4.9.3.2 goBackToMain	54
4.9.3.3 populateDropDown()	54
4.9.3.4 show	55
4.9.3.5 showUserStats	55
4.9.4 Member Data Documentation	55
4.9.4.1 backToMainButton	55
4.9.4.2 gamesPlayedStats	55
4.9.4.3 gamesWinRateStats	55
4.9.4.4 gamesWinStats	55
4.9.4.5 guessHitRateStats	55
4.9.4.6 guessHitStats	56
4.9.4.7 guessTotalStats	56
4.9.4.8 showUserStatsButton	56
4.9.4.9 username	56
4.9.4.10 usernameComboBox	56
4.9.4.11 usernameTitle	56
4.9.4.12 users	56
4.10 Transition Class Reference	57
4.10.1 Detailed Description	58
4.10.2 Constructor & Destructor Documentation	58
4.10.2.1 Transition()	58
4.10.2.2 ~Transition()	58
4.10.3 Member Function Documentation	58
4.10.3.1 continueClicked	58
4.10.3.2 setMessage()	58
4.10.4 Member Data Documentation	59
4.10.4.1 continueButton	59
4.10.4.2 messageLabel	59
4.11 Tutorial Class Reference	59
4.11.1 Detailed Description	61
4.11.2 Constructor & Destructor Documentation	61
4.11.2.1 Tutorial()	61
4.11.2.2 ~Tutorial()	61
4.11.3 Member Function Documentation	61
4.11.3.1 closeEvent()	61
4.11.3.2 onContinueClicked	62
4.11.3.3 resetTutorial()	62
4.11.3.4 tutorialClosed	62
4.11.3.5 updateContinueButtonPosition()	62
4.11.4 Member Data Documentation	62
4.11.4.1 centralWidget	62
4.11.4.2 clickCount	62



4.11.4.3 continueButton . . . . .	63
4.11.4.4 textBox . . . . .	63
4.11.4.5 titleLabel . . . . .	63
4.12 User Class Reference . . . . .	63
4.12.1 Detailed Description . . . . .	66
4.12.2 Constructor & Destructor Documentation . . . . .	66
4.12.2.1 ~User() . . . . .	66
4.12.2.2 User() . . . . .	66
4.12.3 Member Function Documentation . . . . .	66
4.12.3.1 backToMainMenu . . . . .	66
4.12.3.2 getGamesPlayed() . . . . .	66
4.12.3.3 getGuessHit() . . . . .	67
4.12.3.4 getGuessTotal() . . . . .	67
4.12.3.5 getHitRate() . . . . .	67
4.12.3.6 getWinRate() . . . . .	68
4.12.3.7 getWins() . . . . .	68
4.12.3.8 handleCreateAccount . . . . .	68
4.12.3.9 handleLogin . . . . .	68
4.12.3.10 hit() . . . . .	68
4.12.3.11 instance() . . . . .	69
4.12.3.12 loadJsonFile() . . . . .	69
4.12.3.13 lost() . . . . .	69
4.12.3.14 miss() . . . . .	69
4.12.3.15 populateUsernameComboBox() . . . . .	70
4.12.3.16 refreshUserDropdown . . . . .	70
4.12.3.17 renameUser() . . . . .	70
4.12.3.18 show . . . . .	70
4.12.3.19 showMainMenu . . . . .	70
4.12.3.20 updateGamesPlayed() . . . . .	70
4.12.3.21 updateGuessHit() . . . . .	71
4.12.3.22 updateGuessTotal() . . . . .	71
4.12.3.23 updateWins() . . . . .	71
4.12.3.24 won() . . . . .	71
4.12.4 Member Data Documentation . . . . .	72
4.12.4.1 backButton . . . . .	72
4.12.4.2 createAccountButton . . . . .	72
4.12.4.3 createAccountWindow . . . . .	72
4.12.4.4 jsonContentLabel . . . . .	72
4.12.4.5 jsonFilePath . . . . .	72
4.12.4.6 loginButton . . . . .	72
4.12.4.7 usernameComboBox . . . . .	72

<b>5 File Documentation</b>	<b>73</b>
5.1 include/chatbox.h File Reference	73
5.1.1 Detailed Description	74
5.2 chatbox.h	75
5.3 include/createaccountwindow.h File Reference	75
5.3.1 Detailed Description	76
5.4 createaccountwindow.h	77
5.5 include/gameboard.h File Reference	77
5.5.1 Detailed Description	78
5.6 gameboard.h	79
5.7 include/mainwindow.h File Reference	80
5.7.1 Detailed Description	81
5.8 mainwindow.h	81
5.9 include/operatorguess.h File Reference	82
5.9.1 Detailed Description	83
5.10 operatorguess.h	83
5.11 include/pregame.h File Reference	84
5.11.1 Detailed Description	85
5.12 pregame.h	86
5.13 include/spymasterhint.h File Reference	87
5.13.1 Detailed Description	88
5.14 spymasterhint.h	88
5.15 include/statisticswindow.h File Reference	89
5.15.1 Detailed Description	89
5.16 statisticswindow.h	90
5.17 include/transition.h File Reference	91
5.17.1 Detailed Description	92
5.18 transition.h	93
5.19 include/tutorial.h File Reference	93
5.19.1 Detailed Description	94
5.20 tutorial.h	94
5.21 include/user.h File Reference	95
5.21.1 Detailed Description	96
5.22 user.h	97
5.23 src/chatbox.cpp File Reference	98
5.24 src/createaccountwindow.cpp File Reference	98
5.24.1 Detailed Description	99
5.25 src/gameboard.cpp File Reference	99
5.26 src/main.cpp File Reference	99
5.26.1 Function Documentation	100
5.26.1.1 main()	100
5.27 src/mainwindow.cpp File Reference	100

---

5.28 src/operatorguess.cpp File Reference . . . . .	100
5.29 src/pregame.cpp File Reference . . . . .	100
5.29.1 Detailed Description . . . . .	101
5.30 src/spymasterhint.cpp File Reference . . . . .	101
5.31 src/statisticswindow.cpp File Reference . . . . .	101
5.31.1 Detailed Description . . . . .	102
5.32 src/transition.cpp File Reference . . . . .	102
5.33 src/tutorial.cpp File Reference . . . . .	102
5.34 src/user.cpp File Reference . . . . .	103
5.34.1 Detailed Description . . . . .	103
<b>Index</b>	<b>105</b>



# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

GameBoard::Card . . . . .	7
QMainWindow	
MainWindow . . . . .	32
Tutorial . . . . .	59
QWidget	
ChatBox . . . . .	8
CreateAccountWindow . . . . .	13
GameBoard . . . . .	18
OperatorGuess . . . . .	37
PreGame . . . . .	40
SpymasterHint . . . . .	48
StatisticsWindow . . . . .	52
Transition . . . . .	57
User . . . . .	63



## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">GameBoard::Card</a>	Structure representing a card in the game grid . . . . .	7
<a href="#">ChatBox</a>	A widget for the chat feature in the game . . . . .	8
<a href="#">CreateAccountWindow</a>	Singleton interface for creating new user accounts This window allows users to input a username and creates a profile JSON file for the new account . . . . .	13
<a href="#">GameBoard</a>	A class representing the game board for the Spy Master game . . . . .	18
<a href="#">MainWindow</a>	The main application window . . . . .	32
<a href="#">OperatorGuess</a>	A widget that provides the interface for operators to submit guesses during gameplay . . . . .	37
<a href="#">PreGame</a>	Interface for setting up a new game This includes selecting players for each team and role before starting the game . . . . .	40
<a href="#">SpymasterHint</a>	A widget for the spymaster to input a hint and the number of words associated with it . . . . .	48
<a href="#">StatisticsWindow</a>	The class that shows the Statistics screen Displays game statistics for selected users including win rates and guess accuracy . . . . .	52
<a href="#">Transition</a>	A widget for displaying a transition message and a button to continue . . . . .	57
<a href="#">Tutorial</a>	The tutorial window that guides users through the game mechanics . . . . .	59
<a href="#">User</a>	<a href="#">User</a> class to handle local log in and loading/storing json files. This is a singleton class to ensure only one instance of user management exists. Manages user profiles, statistics, and authentication . . . . .	63





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

include/ <a href="#">chatbox.h</a>	Header file for the <a href="#">ChatBox</a> class, which provides a UI for the chat feature in the game . . . . .	73
include/ <a href="#">createaccountwindow.h</a>	Header file for the <a href="#">CreateAccountWindow</a> class which handles user account creation . . . . .	75
include/ <a href="#">gameboard.h</a>	Header file for the <a href="#">GameBoard</a> class, which implements a game board for the Spy Master game . . . . .	77
include/ <a href="#">mainwindow.h</a>	Declaration of the <a href="#">MainWindow</a> class . . . . .	80
include/ <a href="#">operatorguess.h</a>	Header file for the <a href="#">OperatorGuess</a> class, which handles operator guessing interface . . . . .	82
include/ <a href="#">pregame.h</a>	Header file for the <a href="#">PreGame</a> class which handles the game setup screen . . . . .	84
include/ <a href="#">spymasterhint.h</a>	Header file for the <a href="#">SpymasterHint</a> class, which provides a UI for the spymaster to give hints . . . . .	87
include/ <a href="#">statisticswindow.h</a>	The screen to show the user's statistics . . . . .	89
include/ <a href="#">transition.h</a>	Header file for the <a href="#">Transition</a> class, which provides a UI for transitions between game states . . . . .	91
include/ <a href="#">tutorial.h</a>	Declaration of the <a href="#">Tutorial</a> class . . . . .	93
include/ <a href="#">user.h</a>	<a href="#">User</a> class to handle local log in and loading/storing json files . . . . .	95
src/ <a href="#">chatbox.cpp</a>	. . . . .	98
src/ <a href="#">createaccountwindow.cpp</a>	CPP file for the <a href="#">CreateAccountWindow</a> class which handles user account creation . . . . .	98
src/ <a href="#">gameboard.cpp</a>	. . . . .	99
src/ <a href="#">main.cpp</a>	. . . . .	99
src/ <a href="#">mainwindow.cpp</a>	. . . . .	100
src/ <a href="#">operatorguess.cpp</a>	. . . . .	100
src/ <a href="#">pregame.cpp</a>	CPP file for the <a href="#">PreGame</a> class which handles the game setup screen . . . . .	100
src/ <a href="#">spymasterhint.cpp</a>	. . . . .	101
src/ <a href="#">statisticswindow.cpp</a>	The screen to show the user's statistics . . . . .	101
src/ <a href="#">transition.cpp</a>	. . . . .	102
src/ <a href="#">tutorial.cpp</a>	. . . . .	102
src/ <a href="#">user.cpp</a>	<a href="#">User</a> class to handle local log in and loading/storing json files . . . . .	103



## Chapter 4

# Class Documentation

### 4.1 GameBoard::Card Struct Reference

Structure representing a card in the game grid.

#### Public Attributes

- QString [word](#)
- [CardType](#) type
- bool [revealed](#)

#### 4.1.1 Detailed Description

Structure representing a card in the game grid.

Contains the word, type, and revealed status of the card.

#### 4.1.2 Member Data Documentation

##### 4.1.2.1 revealed

```
bool GameBoard::Card::revealed
```

##### 4.1.2.2 type

```
CardType GameBoard::Card::type
```

##### 4.1.2.3 word

```
QString GameBoard::Card::word
```

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

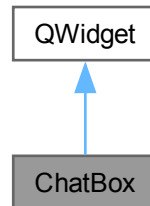
- include/[gameboard.h](#)

## 4.2 ChatBox Class Reference

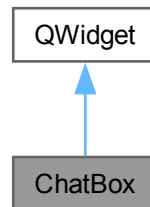
A widget for the chat feature in the game.

```
#include <chatbox.h>
```

Inheritance diagram for ChatBox:



Collaboration diagram for ChatBox:



### Public Types

- enum `Team` { `RED_TEAM` , `BLUE_TEAM` }  
*Enumeration for the two teams in the game.*

### Public Slots

- void `sendMessage` ()  
*Sends a message from the chat input.*

### Signals

- void `massSend` (const QString &`playerName`, const QString &`message`)  
*Signal emitted when a message is sent.*

### Public Member Functions

- [ChatBox](#) (const QString &[playerName](#), [Team](#) [team](#), QWidget \*parent=nullptr)  
*Constructor for the [ChatBox](#) class.*
- [~ChatBox](#) ()  
*Destructor for the [ChatBox](#) class.*
- void [addSystemMessage](#) (const QString &message, [Team](#) [team](#))  
*Adds a system message to the chat display.*
- void [addPlayerMessage](#) (const QString &[playerName](#), const QString &message)  
*Adds a player message to the chat display.*
- void [setPlayerName](#) (const QString &name)  
*Sets the player name for the chat box.*
- void [clearChat](#) ()  
*Clears the chat display.*
- void [limitReachedMessage](#) ()  
*Displays a message when the guess limit is reached.*

### Private Attributes

- [Team](#) [team](#)  
*The team of the player using this chat box.*
- QTextEdit \* [chatDisplay](#)  
*The text edit widget for displaying chat messages.*
- QLineEdit \* [chatInput](#)  
*The line edit widget for inputting chat messages.*
- QPushButton \* [sendButton](#)  
*The button to send chat messages.*
- QString [playerName](#)  
*The name of the player using this chat box.*

## 4.2.1 Detailed Description

A widget for the chat feature in the game.

This class contains a QTextEdit for displaying chat messages, a QLineEdit for inputting messages, and a QPushButton to send messages. It allows players to communicate with each other during the game. It also includes functionality to display system messages and player messages with different styles based on operative guesses and spymaster hints for each team.

### Author

Group 9

## 4.2.2 Member Enumeration Documentation

### 4.2.2.1 Team

```
enum ChatBox::Team
```

Enumeration for the two teams in the game.

This enum is used to differentiate between the two teams (red and blue) in the game. It is used to style the chat messages and system messages based on the team.

## Enumerator

RED_TEAM	
BLUE_TEAM	

## 4.2.3 Constructor & Destructor Documentation

### 4.2.3.1 ChatBox()

```
ChatBox::ChatBox (
    const QString & playerName,
    Team team,
    QWidget * parent = nullptr) [explicit]
```

Constructor for the [ChatBox](#) class.

This constructor sets up the layout and initializes the widgets. It connects the button to the sendMessage slot and the LineEdit to the sendMessage slot.

## Parameters

<i>playerName</i>	The name of the player using this chat box.
<i>team</i>	The team of the player (red or blue).
<i>parent</i>	The parent widget.

### 4.2.3.2 ~ChatBox()

```
ChatBox::~ChatBox ()
```

Destructor for the [ChatBox](#) class.

This destructor cleans up the resources used by the class. It does not need to explicitly delete the widgets as they are managed by Qt's parent-child system.

## 4.2.4 Member Function Documentation

### 4.2.4.1 addPlayerMessage()

```
void ChatBox::addPlayerMessage (
    const QString & playerName,
    const QString & message)
```

Adds a player message to the chat display.

This function adds a player message to the chat box for both local play and online play.

## Parameters

<i>playerName</i>	The name of the player sending the message.
<i>message</i>	The message text.

**4.2.4.2 addSystemMessage()**

```
void ChatBox::addSystemMessage (
    const QString & message,
    Team team)
```

Adds a system message to the chat display.

This function adds a system message to the chat box, printing the operative guesses and spymaster hints for each team. It styles the message based on the team and the type of message.

## Parameters

<i>message</i>	The system message text.
<i>team</i>	The team associated with the message (red or blue).

**4.2.4.3 clearChat()**

```
void ChatBox::clearChat ()
```

Clears the chat display.

This function clears all messages from the chat display so the chat is empty for new games.

**4.2.4.4 limitReachedMessage()**

```
void ChatBox::limitReachedMessage ()
```

Displays a message when the guess limit is reached.

This function displays a message indicating that the operative has reached the limit for their guesses, meaning they cannot make any more guesses and must end their turn.

**4.2.4.5 massSend**

```
void ChatBox::massSend (
    const QString & playerName,
    const QString & message) [signal]
```

Signal emitted when a message is sent.

This signal is emitted when the user sends a message from the chat input field. It carries the player name and the message text as parameters.

**Parameters**

<i>playerName</i>	The name of the player sending the message.
<i>message</i>	The message text.

**4.2.4.6 sendMessage**

```
void ChatBox::sendMessage () [slot]
```

Sends a message from the chat input.

This function retrieves the text from the chat input field and emits a signal to send the message. It also clears the input field after sending the message.

**4.2.4.7 setPlayerName()**

```
void ChatBox::setPlayerName (  
    const QString & name)
```

Sets the player name for the chat box.

This function sets the player name for the chat box, which is used to identify the sender of messages.

**Parameters**

<i>name</i>	The name of the player.
-------------	-------------------------

**4.2.5 Member Data Documentation****4.2.5.1 chatDisplay**

```
QTextEdit* ChatBox::chatDisplay [private]
```

The text edit widget for displaying chat messages.

**4.2.5.2 chatInput**

```
QLineEdit* ChatBox::chatInput [private]
```

The line edit widget for inputting chat messages.

**4.2.5.3 playerName**

```
QString ChatBox::playerName [private]
```

The name of the player using this chat box.



#### 4.2.5.4 sendButton

```
QPushButton* ChatBox::sendButton [private]
```

The button to send chat messages.

#### 4.2.5.5 team

```
Team ChatBox::team [private]
```

The team of the player using this chat box.

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

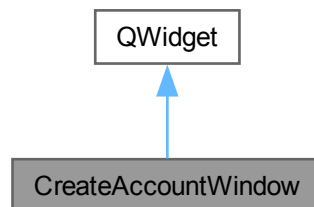
- [include/chatbox.h](#)
- [src/chatbox.cpp](#)

## 4.3 CreateAccountWindow Class Reference

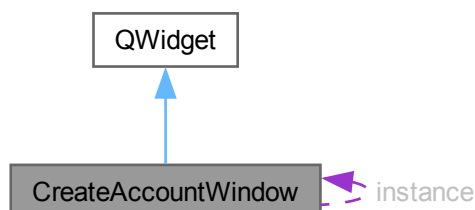
The [CreateAccountWindow](#) class provides a singleton interface for creating new user accounts This window allows users to input a username and creates a profile JSON file for the new account.

```
#include <createaccountwindow.h>
```

Inheritance diagram for CreateAccountWindow:



Collaboration diagram for CreateAccountWindow:



## Public Slots

- void [show](#) ()  
*Displays the account creation window and prepares the UI Resets status messages and input fields when shown.*

## Signals

- void [back](#) ()  
*Signal emitted when returning to the previous screen Connected to the appropriate handler in the previous screen.*
- void [accountCreated](#) ()  
*Signal emitted when a new account is successfully created Notifies other components to update their user lists.*

## Public Member Functions

- void [setPreviousScreen](#) (QWidget \*previous)  
*Set the previous screen to return to when operation is complete Used for navigation back to the calling screen.*

## Static Public Member Functions

- static [CreateAccountWindow](#) \* [getInstance](#) (QWidget \*parent=nullptr)  
*Get the singleton instance of [CreateAccountWindow](#) Creates the instance if it doesn't exist yet.*

## Private Slots

- void [onCreateAccountClicked](#) ()  
*Handles the create account button click event Validates input and creates a new user profile if valid.*
- void [goBack](#) ()  
*Returns to the previous screen Called when account creation is complete or canceled.*

## Private Member Functions

- [CreateAccountWindow](#) (QWidget \*parent=nullptr)  
*Private constructor to enforce singleton pattern Initializes UI components for account creation.*
- void [saveJsonFile](#) (const QString &username)  
*Creates and saves a JSON profile file for the new user Stores basic user information in the specified JSON file.*

## Private Attributes

- QLineEdit \* [usernameEdit](#)  
*Text input field for entering the new username.*
- QPushButton \* [createAccountButton](#)  
*Button to submit account creation request.*
- QLabel \* [statusLabel](#)  
*Label to display status messages and error feedback.*
- QString [jsonFilePath](#) = "resources/profile.json"  
*Path to the JSON profile file where user data will be stored May need to be updated based on deployment environment.*
- QWidget \* [previousScreen](#) = nullptr  
*Pointer to the previous screen to return to after account creation Set via [setPreviousScreen\(\)](#) method.*

## Static Private Attributes

- static [CreateAccountWindow](#) \* *instance* = nullptr

*Static pointer to the singleton instance Ensures only one instance exists throughout the application.*

## 4.3.1 Detailed Description

The [CreateAccountWindow](#) class provides a singleton interface for creating new user accounts This window allows users to input a username and creates a profile JSON file for the new account.

## 4.3.2 Constructor & Destructor Documentation

### 4.3.2.1 CreateAccountWindow()

```
CreateAccountWindow::CreateAccountWindow (
    QWidget * parent = nullptr) [explicit], [private]
```

Private constructor to enforce singleton pattern Initializes UI components for account creation.

#### Parameters

<i>parent</i>	Optional parent widget for memory management
---------------	--

## 4.3.3 Member Function Documentation

### 4.3.3.1 accountCreated

```
void CreateAccountWindow::accountCreated () [signal]
```

Signal emitted when a new account is successfully created Notifies other components to update their user lists.

### 4.3.3.2 back

```
void CreateAccountWindow::back () [signal]
```

Signal emitted when returning to the previous screen Connected to the appropriate handler in the previous screen.

### 4.3.3.3 getInstance()

```
CreateAccountWindow * CreateAccountWindow::getInstance (
    QWidget * parent = nullptr) [static]
```

Get the singleton instance of [CreateAccountWindow](#) Creates the instance if it doesn't exist yet.

**Parameters**

<i>parent</i>	Optional parent widget for memory management purposes
---------------	---

**Returns**

CreateAccountWindow\* Pointer to the singleton instance

**4.3.3.4 goBack**

```
void CreateAccountWindow::goBack () [private], [slot]
```

Returns to the previous screen Called when account creation is complete or canceled.

**4.3.3.5 onCreateAccountClicked**

```
void CreateAccountWindow::onCreateAccountClicked () [private], [slot]
```

Handles the create account button click event Validates input and creates a new user profile if valid.

**4.3.3.6 saveJsonFile()**

```
void CreateAccountWindow::saveJsonFile (  
    const QString & username) [private]
```

Creates and saves a JSON profile file for the new user Stores basic user information in the specified JSON file.

**Parameters**

<i>username</i>	The username for the new account
-----------------	----------------------------------

**4.3.3.7 setPreviousScreen()**

```
void CreateAccountWindow::setPreviousScreen (  
    QWidget * previous)
```

Set the previous screen to return to when operation is complete Used for navigation back to the calling screen.

**Parameters**

<i>previous</i>	Pointer to the widget to return to
-----------------	------------------------------------

**4.3.3.8 show**

```
void CreateAccountWindow::show () [slot]
```

Displays the account creation window and prepares the UI Resets status messages and input fields when shown.

### 4.3.4 Member Data Documentation

#### 4.3.4.1 createAccountButton

```
QPushButton* CreateAccountWindow::createAccountButton [private]
```

Button to submit account creation request.

#### 4.3.4.2 instance

```
CreateAccountWindow * CreateAccountWindow::instance = nullptr [static], [private]
```

Static pointer to the singleton instance Ensures only one instance exists throughout the application.

#### 4.3.4.3 jsonFilePath

```
QString CreateAccountWindow::jsonFilePath = "resources/profile.json" [private]
```

Path to the JSON profile file where user data will be stored May need to be updated based on deployment environment.

#### 4.3.4.4 previousScreen

```
QWidget* CreateAccountWindow::previousScreen = nullptr [private]
```

Pointer to the previous screen to return to after account creation Set via [setPreviousScreen\(\)](#) method.

#### 4.3.4.5 statusLabel

```
QLabel* CreateAccountWindow::statusLabel [private]
```

Label to display status messages and error feedback.

#### 4.3.4.6 usernameEdit

```
QLineEdit* CreateAccountWindow::usernameEdit [private]
```

Text input field for entering the new username.

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

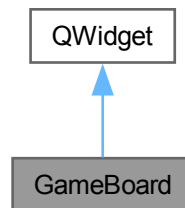
- [include/createaccountwindow.h](#)
- [src/createaccountwindow.cpp](#)

## 4.4 GameBoard Class Reference

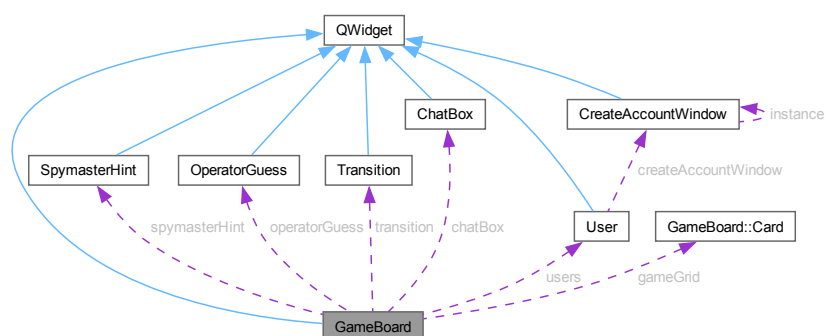
A class representing the game board for the Spy Master game.

```
#include <gameboard.h>
```

Inheritance diagram for GameBoard:



Collaboration diagram for GameBoard:



### Classes

- struct [Card](#)  
*Structure representing a card in the game grid.*

### Public Slots

- void [show](#) ()  
*Displays the game board.*
- void [displayHint](#) (const QString &hint, int number)  
*Displays a hint on the game board.*
- void [displayGuess](#) ()  
*Displays a guess on the game board.*

## Signals

- void [gameEnded](#) ()  
*Emitted when the game ends.*

## Public Member Functions

- [GameBoard](#) (const QString &redSpyMaster, const QString &redOperative, const QString &blueSpyMaster, const QString &blueOperative, QWidget \*parent=nullptr)  
*Constructor for the [GameBoard](#) class.*
- [~GameBoard](#) ()  
*Destructor for the [GameBoard](#) class.*
- void [setRedSpyMasterName](#) (const QString &name)  
*Sets the names of the red team's spymaster and operative.*
- void [setRedOperativeName](#) (const QString &name)  
*Sets the names of the red team's operative.*
- void [setBlueSpyMasterName](#) (const QString &name)  
*Sets the names of the blue team's spymaster and operative.*
- void [setBlueOperativeName](#) (const QString &name)  
*Sets the names of the blue team's operative.*
- void [updateTeamLabels](#) ()  
*Updates the labels displaying team information.*

## Private Types

- enum [CardType](#) { [RED\\_TEAM](#) , [BLUE\\_TEAM](#) , [NEUTRAL](#) , [ASSASSIN](#) }  
*Enumeration for card types.*
- enum [Turn](#) { [RED\\_SPY](#) , [RED\\_OP](#) , [BLUE\\_SPY](#) , [BLUE\\_OP](#) }  
*Enumeration representing the different turn states in the game board.*

## Private Member Functions

- void [loadWordsFromFile](#) ()  
*Loads words from a file and generates the game grid.*
- void [generateGameGrid](#) ()  
*Generates the game grid.*
- void [setupUI](#) ()  
*Sets up the UI for the game board.*
- void [nextTurn](#) ()  
*Switches to the next turn.*
- void [onCardClicked](#) (int row, int col)  
*Handles a card click event.*
- void [onContinueClicked](#) ()  
*Handles the continue button click event.*
- void [showTransition](#) ()  
*Displays a transition screen.*
- void [updateScores](#) ()  
*Updates the scores of the teams.*
- void [checkGameEnd](#) ()  
*Checks if the game has ended.*
- void [endGame](#) (const QString &message)  
*Ends the game and displays a message.*
- void [resetGame](#) ()  
*Resets the game state.*

## Private Attributes

- int `currentTurn`  
*Structure representing a turn in the game board.*
- int `redCardsRemaining`  
*The number of remaining cards for each team.*
- int `blueCardsRemaining`  
*The number of remaining cards for each team.*
- int `maxGuesses` = 0  
*The maximum number of guesses allowed in a turn.*
- int `currentGuesses` = 0  
*The number of guesses made in the current turn.*
- QString `redSpyMasterName`  
*The names of the spymaster for the red team.*
- QString `redOperativeName`  
*The names of the operative for the blue team.*
- QString `blueSpyMasterName`  
*The names of the spymaster for the blue team.*
- QString `blueOperativeName`  
*The names of the operative for the blue team.*
- Card gameGrid [GRID\_SIZE][GRID\_SIZE]  
*The game grid.*
- QStringList `wordList`  
*The list of words.*
- QGridLayout \* `gridLayout`  
*The grid layout for the game board.*
- QPushButton \* `cards` [GRID\_SIZE][GRID\_SIZE]  
*The buttons representing the cards in the game grid.*
- QLabel \* `redTeamLabel`  
*The labels for red team information.*
- QLabel \* `blueTeamLabel`  
*The label for blue team information.*
- QLabel \* `currentTurnLabel`  
*The label for the current turn.*
- SpymasterHint \* `spymasterHint`  
*The widget for the spymaster hint.*
- OperatorGuess \* `operatorGuess`  
*The widget for the operator guess.*
- QLabel \* `currentHint`  
*The label for the current hint.*
- QString `correspondingNumber`  
*The label for the corresponding number.*
- Transition \* `transition`  
*The transition screen widget.*
- QLabel \* `redScoreLabel`  
*The label for red team score.*
- QLabel \* `blueScoreLabel`  
*The label for blue team score.*
- ChatBox \* `chatBox`  
*The chat box widget.*
- QString `currentPlayerName`



*The name of the current player.*

- `ChatBox::Team currentPlayerTeam`

*The team of the current player.*

- `User * users`

*The list of users in the game.*

### Static Private Attributes

- static const int `GRID_SIZE` = 5

*The size of the game grid.*

## 4.4.1 Detailed Description

A class representing the game board for the Spy Master game.

The `GameBoard` class is responsible for displaying the game board and handling user interactions. It includes methods for loading words from a file, generating the game grid, setting up the UI, card clicks, card reveals and turns, and handling game end conditions. The game board also includes a stacked layout for transitions between screens. Codenames is a game which involves two teams (red and blue) with spymasters giving hints and operators making guesses.

Author

Group 9

## 4.4.2 Member Enumeration Documentation

### 4.4.2.1 CardType

```
enum GameBoard::CardType [private]
```

Enumeration for card types.

Enumerator

RED_TEAM	
BLUE_TEAM	
NEUTRAL	
ASSASSIN	

### 4.4.2.2 Turn

```
enum GameBoard::Turn [private]
```

Enumeration representing the different turn states in the game board.

### Enumerator

RED_SPY	
RED_OP	
BLUE_SPY	
BLUE_OP	

## 4.4.3 Constructor & Destructor Documentation

### 4.4.3.1 GameBoard()

```
GameBoard::GameBoard (
    const QString & redSpyMaster,
    const QString & redOperative,
    const QString & blueSpyMaster,
    const QString & blueOperative,
    QWidget * parent = nullptr) [explicit]
```

Constructor for the [GameBoard](#) class.

Initializes the game board with the provided team names and sets up the UI. It also loads words from a file and generates the game grid which is displayed on the UI.

#### Parameters

<i>redSpyMaster</i>	The name of the red team's spymaster.
<i>redOperative</i>	The name of the red team's operative.
<i>blueSpyMaster</i>	The name of the blue team's spymaster.
<i>blueOperative</i>	The name of the blue team's operative.
<i>parent</i>	Optional parent widget.

#### Author

Group 9

### 4.4.3.2 ~GameBoard()

```
GameBoard::~GameBoard ()
```

Destructor for the [GameBoard](#) class.

Cleans up resources used by the game board.

#### Author

Group 9

## 4.4.4 Member Function Documentation

### 4.4.4.1 checkGameEnd()

```
void GameBoard::checkGameEnd () [private]
```

Checks if the game has ended.

Checks if the game has ended based on the current state of the game.

Author

Group 9

### 4.4.4.2 displayGuess

```
void GameBoard::displayGuess () [slot]
```

Displays a guess on the game board.

Displays a guess on the game board for the current turn and updates the UI.

Author

Group 9

### 4.4.4.3 displayHint

```
void GameBoard::displayHint (
    const QString & hint,
    int number) [slot]
```

Displays a hint on the game board.

Displays a hint on the game board for the current turn and updates the UI.

Parameters

<i>hint</i>	The hint to be displayed.
<i>number</i>	The number of words associated with the hint.

Author

Group 9

### 4.4.4.4 endGame()

```
void GameBoard::endGame (
    const QString & message) [private]
```

Ends the game and displays a message.

Ends the game and displays a message.

**Parameters**

<i>message</i>	The message to be displayed.
----------------	------------------------------

**Author**

Group 9

**4.4.4.5 gameEnded**

```
void GameBoard::gameEnded () [signal]
```

Emitted when the game ends.

Signals that the game has ended and the game board should be closed.

**Author**

Group 9

**4.4.4.6 generateGameGrid()**

```
void GameBoard::generateGameGrid () [private]
```

Generates the game grid.

Generates the game grid based on the loaded words.

**Author**

Group 9

**4.4.4.7 loadWordsFromFile()**

```
void GameBoard::loadWordsFromFile () [private]
```

Loads words from a file and generates the game grid.

Loads words from a file and generates the game grid.

**Author**

Group 9

#### 4.4.4.8 nextTurn()

```
void GameBoard::nextTurn () [private]
```

Switches to the next turn.

Switches to the next turn and updates the UI.

Author

Group 9

#### 4.4.4.9 onCardClicked()

```
void GameBoard::onCardClicked (
    int row,
    int col) [private]
```

Handles a card click event.

Handles a card click event and updates the UI.

Parameters

<i>row</i>	The row of the clicked card.
<i>col</i>	The column of the clicked card.

Author

Group 9

#### 4.4.4.10 onContinueClicked()

```
void GameBoard::onContinueClicked () [private]
```

Handles the continue button click event.

Handles the continue button click event and updates the UI.

Author

Group 9

#### 4.4.4.11 resetGame()

```
void GameBoard::resetGame () [private]
```

Resets the game state.

Resets the game state to the initial state.

Author

Group 9

#### 4.4.4.12 setBlueOperativeName()

```
void GameBoard::setBlueOperativeName (
    const QString & name)
```

Sets the names of the blue team's operative.

Sets the names of the blue team's operative and updates the team labels.

**Parameters**

<i>name</i>	The name of the blue team's operative.
-------------	--

**Author**

Group 9

**4.4.4.13 setBlueSpyMasterName()**

```
void GameBoard::setBlueSpyMasterName (  
    const QString & name)
```

Sets the names of the blue team's spymaster and operative.

Sets the names of the blue team's spymaster and operative and updates the team labels.

**Parameters**

<i>name</i>	The name of the blue team's spymaster.
-------------	--

**Author**

Group 9

**4.4.4.14 setRedOperativeName()**

```
void GameBoard::setRedOperativeName (  
    const QString & name)
```

Sets the names of the red team's operative.

Sets the names of the red team's operative and updates the team labels.

**Parameters**

<i>name</i>	The name of the red team's operative.
-------------	---------------------------------------

**Author**

Group 9

**4.4.4.15 setRedSpyMasterName()**

```
void GameBoard::setRedSpyMasterName (  
    const QString & name)
```

Sets the names of the red team's spymaster and operative.

Sets the names of the red team's spymaster and operative.

## Parameters

<i>name</i>	The name of the red team's spymaster.
-------------	---------------------------------------

## Author

Group 9

**4.4.4.16 setupUI()**

```
void GameBoard::setupUI () [private]
```

Sets up the UI for the game board.

Sets up the UI for the game board, including the layout, labels, and buttons.

## Author

Group 9

**4.4.4.17 show**

```
void GameBoard::show () [slot]
```

Displays the game board.

Displays the game board and sets up the UI.

## Author

Group 9

**4.4.4.18 showTransition()**

```
void GameBoard::showTransition () [private]
```

Displays a transition screen.

Displays a transition screen and updates the UI.

## Author

Group 9

#### 4.4.4.19 updateScores()

```
void GameBoard::updateScores () [private]
```

Updates the scores of the teams.

Updates the scores of the teams based on the current state of the game.

Author

Group 9

#### 4.4.4.20 updateTeamLabels()

```
void GameBoard::updateTeamLabels ()
```

Updates the labels displaying team information.

Updates the labels displaying team information, such as team names and scores.

Author

Group 9

### 4.4.5 Member Data Documentation

#### 4.4.5.1 blueCardsRemaining

```
int GameBoard::blueCardsRemaining [private]
```

The number of remaining cards for each team.

#### 4.4.5.2 blueOperativeName

```
QString GameBoard::blueOperativeName [private]
```

The names of the operative for the blue team.

#### 4.4.5.3 blueScoreLabel

```
QLabel* GameBoard::blueScoreLabel [private]
```

The label for blue team score.

#### 4.4.5.4 blueSpyMasterName

```
QString GameBoard::blueSpyMasterName [private]
```

The names of the spymaster for the blue team.



#### 4.4.5.5 blueTeamLabel

```
QLabel* GameBoard::blueTeamLabel [private]
```

The label for blue team information.

#### 4.4.5.6 cards

```
QPushButton* GameBoard::cards[GRID_SIZE][GRID_SIZE] [private]
```

The buttons representing the cards in the game grid.

#### 4.4.5.7 chatBox

```
ChatBox* GameBoard::chatBox [private]
```

The chat box widget.

#### 4.4.5.8 correspondingNumber

```
QString GameBoard::correspondingNumber [private]
```

The label for the corresponding number.

#### 4.4.5.9 currentGuesses

```
int GameBoard::currentGuesses = 0 [private]
```

The number of guesses made in the current turn.

#### 4.4.5.10 currentHint

```
QLabel* GameBoard::currentHint [private]
```

The label for the current hint.

#### 4.4.5.11 currentPlayerName

```
QString GameBoard::currentPlayerName [private]
```

The name of the current player.

#### 4.4.5.12 currentPlayerTeam

```
ChatBox::Team GameBoard::currentPlayerTeam [private]
```

The team of the current player.

#### 4.4.5.13 currentTurn

```
int GameBoard::currentTurn [private]
```

Structure representing a turn in the game board.

#### 4.4.5.14 currentTurnLabel

```
QLabel* GameBoard::currentTurnLabel [private]
```

The label for the current turn.

#### 4.4.5.15 gameGrid

```
Card GameBoard::gameGrid[GRID_SIZE][GRID_SIZE] [private]
```

The game grid.

#### 4.4.5.16 GRID\_SIZE

```
const int GameBoard::GRID_SIZE = 5 [static], [private]
```

The size of the game grid.

#### 4.4.5.17 gridLayout

```
QGridLayout* GameBoard::gridLayout [private]
```

The grid layout for the game board.

#### 4.4.5.18 maxGuesses

```
int GameBoard::maxGuesses = 0 [private]
```

The maximum number of guesses allowed in a turn.

#### 4.4.5.19 operatorGuess

```
OperatorGuess* GameBoard::operatorGuess [private]
```

The widget for the operator guess.

#### 4.4.5.20 redCardsRemaining

```
int GameBoard::redCardsRemaining [private]
```

The number of remaining cards for each team.

#### 4.4.5.21 redOperativeName

```
QString GameBoard::redOperativeName [private]
```

The names of the operative for the blue team.

#### 4.4.5.22 redScoreLabel

```
QLabel* GameBoard::redScoreLabel [private]
```

The label for red team score.

#### 4.4.5.23 redSpyMasterName

```
QString GameBoard::redSpyMasterName [private]
```

The names of the spymaster for the red team.

#### 4.4.5.24 redTeamLabel

```
QLabel* GameBoard::redTeamLabel [private]
```

The labels for red team information.

#### 4.4.5.25 spymasterHint

```
SpymasterHint* GameBoard::spymasterHint [private]
```

The widget for the spymaster hint.

#### 4.4.5.26 transition

```
Transition* GameBoard::transition [private]
```

The transition screen widget.

#### 4.4.5.27 users

```
User* GameBoard::users [private]
```

The list of users in the game.



### Public Slots

- void [showMainWindow](#) ()  
*Displays the main window.*

### Public Member Functions

- [MainWindow](#) (QWidget \*parent=nullptr)  
*Constructor for [MainWindow](#).*
- [~MainWindow](#) ()  
*Destructor for [MainWindow](#).*

### Private Slots

- void [openPreGame](#) ()  
*Opens the [PreGame](#) window.*
- void [openStatsWindow](#) ()  
*Opens the statistics window.*
- void [openCreateAccount](#) ()  
*Opens the Create Account window.*
- void [openTutorial](#) ()  
*Opens the [Tutorial](#) window.*
- void [openMultiMain](#) ()  
*Opens the Multiplayer main window.*

### Private Attributes

- QWidget \* [centralWidget](#)
- QVBoxLayout \* [layout](#)  
*Layout for organizing the widgets vertically.*
- QLabel \* [titleLabel](#)  
*Label displaying the application title.*
- [PreGame](#) \* [preGameWindow](#)
- QPushButton \* [localPlayButton](#)  
*Button for starting a local game.*
- QPushButton \* [onlinePlayButton](#)  
*Button for starting an online game.*
- QPushButton \* [tutorialButton](#)  
*Button for opening the tutorial.*
- QPushButton \* [statsButton](#)  
*Button for opening the statistics window.*
- QPushButton \* [createAccountButton](#)  
*Button for opening the account creation window.*
- [CreateAccountWindow](#) \* [createAccountWindow](#)  
*Pointer to the account creation window.*
- [StatisticsWindow](#) \* [statsWindow](#)  
*Pointer to the statistics window displaying game stats.*
- [Tutorial](#) \* [tutorialWindow](#)

### 4.5.1 Detailed Description

The main application window.

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 MainWindow()

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

Constructor for [MainWindow](#).

Parameters

<i>parent</i>	The parent widget (default is nullptr).
---------------	---

#### 4.5.2.2 ~MainWindow()

```
MainWindow::~MainWindow ()
```

Destructor for [MainWindow](#).

### 4.5.3 Member Function Documentation

#### 4.5.3.1 openCreateAccount

```
void MainWindow::openCreateAccount () [private], [slot]
```

Opens the Create Account window.

#### 4.5.3.2 openMultiMain

```
void MainWindow::openMultiMain () [private], [slot]
```

Opens the Multiplayer main window.

#### 4.5.3.3 openPreGame

```
void MainWindow::openPreGame () [private], [slot]
```

Opens the [PreGame](#) window.

#### 4.5.3.4 openStatsWindow

```
void MainWindow::openStatsWindow () [private], [slot]
```

Opens the statistics window.

#### 4.5.3.5 openTutorial

```
void MainWindow::openTutorial () [private], [slot]
```

Opens the [Tutorial](#) window.

#### 4.5.3.6 showMainWindow

```
void MainWindow::showMainWindow () [slot]
```

Displays the main window.

### 4.5.4 Member Data Documentation

#### 4.5.4.1 centralWidget

```
QWidget* MainWindow::centralWidget [private]
```

Pointer to the central widget, which holds all main UI elements.

#### 4.5.4.2 createAccountButton

```
QPushButton* MainWindow::createAccountButton [private]
```

Button for opening the account creation window.

#### 4.5.4.3 createAccountWindow

```
CreateAccountWindow* MainWindow::createAccountWindow [private]
```

Pointer to the account creation window.

#### 4.5.4.4 layout

```
QVBoxLayout* MainWindow::layout [private]
```

Layout for organizing the widgets vertically.

#### 4.5.4.5 localPlayButton

```
QPushButton* MainWindow::localPlayButton [private]
```

Button for starting a local game.

#### 4.5.4.6 onlinePlayButton

```
QPushButton* MainWindow::onlinePlayButton [private]
```

Button for starting an online game.

#### 4.5.4.7 preGameWindow

```
PreGame* MainWindow::preGameWindow [private]
```

Pointer to the [PreGame](#) window for local gameplay setup.

#### 4.5.4.8 statsButton

```
QPushButton* MainWindow::statsButton [private]
```

Button for opening the statistics window.

#### 4.5.4.9 statsWindow

```
StatisticsWindow* MainWindow::statsWindow [private]
```

Pointer to the statistics window displaying game stats.

#### 4.5.4.10 titleLabel

```
QLabel* MainWindow::titleLabel [private]
```

Label displaying the application title.

#### 4.5.4.11 tutorialButton

```
QPushButton* MainWindow::tutorialButton [private]
```

Button for opening the tutorial.



#### 4.5.4.12 tutorialWindow

```
Tutorial* MainWindow::tutorialWindow [private]
```

Pointer to the tutorial window explaining the game mechanics.

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

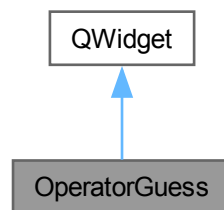
- include/mainwindow.h
- src/mainwindow.cpp

## 4.6 OperatorGuess Class Reference

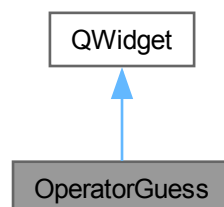
A widget that provides the interface for operators to submit guesses during gameplay.

```
#include <operatorguess.h>
```

Inheritance diagram for OperatorGuess:



Collaboration diagram for OperatorGuess:



## Signals

- void [guessSubmitted](#) ()  
*Signal emitted when a guess is submitted.*

## Public Member Functions

- [OperatorGuess](#) (QWidget \*parent=nullptr)  
*Constructor for the [OperatorGuess](#) class.*
- [~OperatorGuess](#) ()  
*Destructor for the [OperatorGuess](#) class.*
- void [reset](#) ()  
*Resets the operator guess interface.*

## Private Slots

- void [submitGuess](#) ()  
*Handles the submission of a guess.*

## Private Attributes

- QPushButton \* [submitGuessButton](#)  
*Button for submitting a guess.*

## 4.6.1 Detailed Description

A widget that provides the interface for operators to submit guesses during gameplay.

The [OperatorGuess](#) class provides a simple UI for team operators to submit their guesses during their turn. It consists of a button that the operator can click to indicate they have made a guess on the game board.

### Author

Group 9

## 4.6.2 Constructor & Destructor Documentation

### 4.6.2.1 OperatorGuess()

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

Constructor for the [OperatorGuess](#) class.

Initializes the operator guess interface with a submit button. Sets up the UI components and connections.

## Parameters

<i>parent</i>	Optional parent widget.
---------------	-------------------------

## Author

Group 9

#### 4.6.2.2 ~OperatorGuess()

```
OperatorGuess::~~OperatorGuess ()
```

Destructor for the [OperatorGuess](#) class.

Cleans up resources used by the [OperatorGuess](#) widget.

## Author

Group 9

### 4.6.3 Member Function Documentation

#### 4.6.3.1 guessSubmitted

```
void OperatorGuess::guessSubmitted () [signal]
```

Signal emitted when a guess is submitted.

Indicates that the operator has clicked the submit button to register their guess on the game board.

#### 4.6.3.2 reset()

```
void OperatorGuess::reset ()
```

Resets the operator guess interface.

Resets the state of the interface to prepare it for a new turn. This may involve enabling/disabling the button or clearing any internal state.

## Author

Group 9

#### 4.6.3.3 submitGuess

```
void OperatorGuess::submitGuess () [private], [slot]
```

Handles the submission of a guess.

Processes the operator's action when they click the submit button to indicate they have made a guess. Emits the guessSubmitted signal.

Author

Group 9

### 4.6.4 Member Data Documentation

#### 4.6.4.1 submitGuessButton

```
QPushButton* OperatorGuess::submitGuessButton [private]
```

Button for submitting a guess.

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

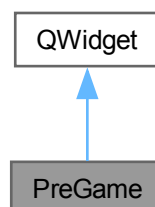
- [include/operatorguess.h](#)
- [src/operatorguess.cpp](#)

## 4.7 PreGame Class Reference

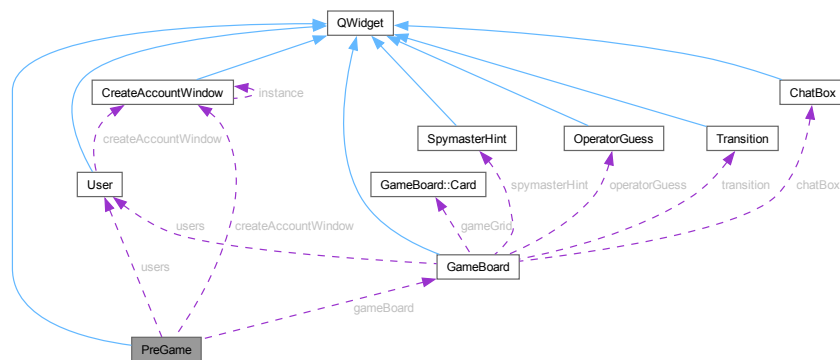
The [PreGame](#) class provides the interface for setting up a new game This includes selecting players for each team and role before starting the game.

```
#include <pregame.h>
```

Inheritance diagram for PreGame:



Collaboration diagram for PreGame:



### Public Slots

- void [show](#) ()  
*Shows the pregame setup window and initializes user dropdowns.*

### Signals

- void [backToMainWindow](#) ()  
*Signal emitted when user wants to return to main window Connected to main window to show it again.*
- void [start](#) ()  
*Signal emitted when all players are selected and game is ready to start Connected to game initialization in the game controller.*
- void [update](#) ()  
*Signal emitted when user list needs to be refreshed This happens after a new account is created.*

### Public Member Functions

- [PreGame](#) (QWidget \*parent=nullptr)  
*Construct a new Pre Game object.*
- [~PreGame](#) ()  
*Destroy the Pre Game object and clean up resources.*
- QString [getRedTeamSpyMasterNickname](#) () const  
*Get the Red Team Spy Master Nickname.*
- QString [getRedTeamOperativeNickname](#) () const  
*Get the Red Team Operative Nickname.*
- QString [getBlueTeamSpyMasterNickname](#) () const  
*Get the Blue Team Spy Master Nickname.*
- QString [getBlueTeamOperativeNickname](#) () const  
*Get the Blue Team Operative Nickname.*

### Private Slots

- void [goBackToMain](#) ()  
*Returns to the main menu screen Connected to the back button's clicked signal.*
- void [startGame](#) ()  
*Starts the game with the selected players Validates player selections and emits start signal if valid.*
- void [handleGameEnd](#) ()  
*Handles cleanup after a game has ended Prepares the UI for a potential new game.*
- void [openCreateAccount](#) ()  
*Opens the account creation window Connected to the create account button's clicked signal.*

### Private Member Functions

- void [populateUserDropdowns](#) ()  
*Populates the user selection dropdown menus with available users This is called when the window is shown to ensure the latest user list.*

### Private Attributes

- [User](#) \* [users](#)  
*Pointer to [User](#) objects containing player information Used to populate the dropdown menus.*
- QStringList [usernames](#)  
*List of available usernames for player selection Populated from the users database.*
- [CreateAccountWindow](#) \* [createAccountWindow](#)  
*Pointer to the account creation window Initialized when create account button is clicked.*
- QLabel \* [label](#)  
*Title label for the pregame screen.*
- QPushButton \* [backButton](#)  
*Button to return to the main menu.*
- QPushButton \* [createAccountButton](#)  
*Button to open the account creation window.*
- QPushButton \* [startButton](#)  
*Button to start the game with selected players.*
- QComboBox \* [redTeamSpyMasterComboBox](#)  
*Dropdown menu for selecting the Red Team's Spy Master.*
- QComboBox \* [redTeamOperativeComboBox](#)  
*Dropdown menu for selecting the Red Team's Operative.*
- QComboBox \* [blueTeamSpyMasterComboBox](#)  
*Dropdown menu for selecting the Blue Team's Spy Master.*
- QComboBox \* [blueTeamOperativeComboBox](#)  
*Dropdown menu for selecting the Blue Team's Operative.*
- QVBoxLayout \* [layout](#)  
*Main vertical layout for the entire pregame screen.*
- QHBoxLayout \* [teamsLayout](#)  
*Horizontal layout to contain both team selection areas.*
- QVBoxLayout \* [redTeamLayout](#)  
*Vertical layout for the Red Team's player selections.*
- QVBoxLayout \* [blueTeamLayout](#)  
*Vertical layout for the Blue Team's player selections.*
- QHBoxLayout \* [buttonsLayout](#)  
*Horizontal layout for the navigation buttons.*
- [GameBoard](#) \* [gameBoard](#)  
*Pointer to the game board that will be shown after game starts.*

### 4.7.1 Detailed Description

The [PreGame](#) class provides the interface for setting up a new game This includes selecting players for each team and role before starting the game.

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 PreGame()

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

Construct a new Pre Game object.

##### Parameters

<i>parent</i>	Optional parent widget for memory management purposes
---------------	---

#### 4.7.2.2 ~PreGame()

```
PreGame::~~PreGame ()
```

Destroy the Pre Game object and clean up resources.

### 4.7.3 Member Function Documentation

#### 4.7.3.1 backToMainWindow

```
void PreGame::backToMainWindow () [signal]
```

Signal emitted when user wants to return to main window Connected to main window to show it again.

#### 4.7.3.2 getBlueTeamOperativeNickname()

```
QString PreGame::getBlueTeamOperativeNickname () const
```

Get the Blue Team Operative Nickname.

##### Returns

QString The nickname of the selected Blue Team Operative

#### 4.7.3.3 getBlueTeamSpyMasterNickname()

```
QString PreGame::getBlueTeamSpyMasterNickname () const
```

Get the Blue Team Spy Master Nickname.

##### Returns

QString The nickname of the selected Blue Team Spy Master

#### 4.7.3.4 getRedTeamOperativeNickname()

```
QString PreGame::getRedTeamOperativeNickname () const
```

Get the Red Team Operative Nickname.

##### Returns

QString The nickname of the selected Red Team Operative

#### 4.7.3.5 getRedTeamSpyMasterNickname()

```
QString PreGame::getRedTeamSpyMasterNickname () const
```

Get the Red Team Spy Master Nickname.

##### Returns

QString The nickname of the selected Red Team Spy Master

#### 4.7.3.6 goBackToMain

```
void PreGame::goBackToMain () [private], [slot]
```

Returns to the main menu screen Connected to the back button's clicked signal.

#### 4.7.3.7 handleGameEnd

```
void PreGame::handleGameEnd () [private], [slot]
```

Handles cleanup after a game has ended Prepares the UI for a potential new game.

#### 4.7.3.8 openCreateAccount

```
void PreGame::openCreateAccount () [private], [slot]
```

Opens the account creation window Connected to the create account button's clicked signal.



#### 4.7.3.9 populateUserDropdowns()

```
void PreGame::populateUserDropdowns () [private]
```

Populates the user selection dropdown menus with available users This is called when the window is shown to ensure the latest user list.

#### 4.7.3.10 show

```
void PreGame::show () [slot]
```

Shows the pregame setup window and initializes user dropdowns.

#### 4.7.3.11 start

```
void PreGame::start () [signal]
```

Signal emitted when all players are selected and game is ready to start Connected to game initialization in the game controller.

#### 4.7.3.12 startGame

```
void PreGame::startGame () [private], [slot]
```

Starts the game with the selected players Validates player selections and emits start signal if valid.

#### 4.7.3.13 update

```
void PreGame::update () [signal]
```

Signal emitted when user list needs to be refreshed This happens after a new account is created.

### 4.7.4 Member Data Documentation

#### 4.7.4.1 backButton

```
QPushButton* PreGame::backButton [private]
```

Button to return to the main menu.

#### 4.7.4.2 blueTeamLayout

```
QVBoxLayout* PreGame::blueTeamLayout [private]
```

Vertical layout for the Blue Team's player selections.

#### 4.7.4.3 blueTeamOperativeComboBox

```
QComboBox* PreGame::blueTeamOperativeComboBox [private]
```

Dropdown menu for selecting the Blue Team's Operative.

#### 4.7.4.4 blueTeamSpyMasterComboBox

```
QComboBox* PreGame::blueTeamSpyMasterComboBox [private]
```

Dropdown menu for selecting the Blue Team's Spy Master.

#### 4.7.4.5 buttonsLayout

```
QHBoxLayout* PreGame::buttonsLayout [private]
```

Horizontal layout for the navigation buttons.

#### 4.7.4.6 createAccountButton

```
QPushButton* PreGame::createAccountButton [private]
```

Button to open the account creation window.

#### 4.7.4.7 createAccountWindow

```
CreateAccountWindow* PreGame::createAccountWindow [private]
```

Pointer to the account creation window Initialized when create account button is clicked.

#### 4.7.4.8 gameBoard

```
GameBoard* PreGame::gameBoard [private]
```

Pointer to the game board that will be shown after game starts.

#### 4.7.4.9 label

```
QLabel* PreGame::label [private]
```

Title label for the pregame screen.

#### 4.7.4.10 layout

```
QVBoxLayout* PreGame::layout [private]
```

Main vertical layout for the entire pregame screen.

#### 4.7.4.11 redTeamLayout

```
QVBoxLayout* PreGame::redTeamLayout [private]
```

Vertical layout for the Red Team's player selections.

#### 4.7.4.12 redTeamOperativeComboBox

```
QComboBox* PreGame::redTeamOperativeComboBox [private]
```

Dropdown menu for selecting the Red Team's Operative.

#### 4.7.4.13 redTeamSpyMasterComboBox

```
QComboBox* PreGame::redTeamSpyMasterComboBox [private]
```

Dropdown menu for selecting the Red Team's Spy Master.

#### 4.7.4.14 startButton

```
QPushButton* PreGame::startButton [private]
```

Button to start the game with selected players.

#### 4.7.4.15 teamsLayout

```
QHBoxLayout* PreGame::teamsLayout [private]
```

Horizontal layout to contain both team selection areas.

#### 4.7.4.16 usernames

```
QStringList PreGame::usernames [private]
```

List of available usernames for player selection Populated from the users database.

#### 4.7.4.17 users

```
User* PreGame::users [private]
```

Pointer to [User](#) objects containing player information Used to populate the dropdown menus.

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

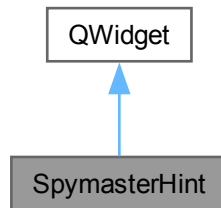
- [include/pregame.h](#)
- [src/pregame.cpp](#)

## 4.8 SpymasterHint Class Reference

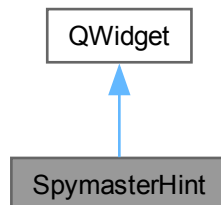
A widget for the spymaster to input a hint and the number of words associated with it.

```
#include <spymasterhint.h>
```

Inheritance diagram for SpymasterHint:



Collaboration diagram for SpymasterHint:



### Signals

- void [hintSubmitted](#) (const QString &hint, const int number)  
*Signal emitted when a hint is submitted.*

### Public Member Functions

- [SpymasterHint](#) (QWidget \*parent=nullptr)  
*Constructor for the [SpymasterHint](#) class.*
- [~SpymasterHint](#) ()  
*Destructor for the [SpymasterHint](#) class.*
- void [reset](#) ()  
*Resets the spymaster hint input fields.*

### Private Slots

- void [submitHint](#) ()  
*Slot to handle the submission of a hint.*
- void [updateButtonClickable](#) ()  
*Slot to update the button's clickable state based on input.*
- void [textToUppercase](#) (const QString &text)  
*Slot to convert text to uppercase.*

### Private Attributes

- QLineEdit \* [hintLineEdit](#)  
*QLineEdit used by the spymaster to input the hint.*
- QSpinBox \* [numberSpinBox](#)  
*QSpinBox used by the spymaster to input the number of words correlated to the hint.*
- QPushButton \* [giveClueButton](#)  
*QPushButton to submit the hint.*
- QRegularExpressionValidator \* [textValidator](#)  
*QRegularExpressionValidator used to validate the hint the spymaster inputs is a single valid word.*

## 4.8.1 Detailed Description

A widget for the spymaster to input a hint and the number of words associated with it.

This class contains a QLineEdit for the hint, a QSpinBox for the number of words, and a QPushButton to submit the hint. It also includes validation to ensure the hint is a single word and updates the button's clickable state based on input.

### Author

Group 9

## 4.8.2 Constructor & Destructor Documentation

### 4.8.2.1 SpymasterHint()

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

Constructor for the [SpymasterHint](#) class.

This constructor sets up the layout and initializes the widgets. It connects the button to the submitHint slot and the LineEdit to the updateButtonClickable slot. It also sets up a validator to ensure the hint is a single word and connects the textChanged signal to the textToUppercase slot to convert the hint to uppercase.

### Parameters

<i>parent</i>	The parent widget.
---------------	--------------------

#### 4.8.2.2 ~SpymasterHint()

```
SpymasterHint::~SpymasterHint ()
```

Destructor for the [SpymasterHint](#) class.

This destructor cleans up the resources used by the class. It does not need to explicitly delete the widgets as they are managed by Qt's parent-child system.

### 4.8.3 Member Function Documentation

#### 4.8.3.1 hintSubmitted

```
void SpymasterHint::hintSubmitted (
    const QString & hint,
    const int number) [signal]
```

Signal emitted when a hint is submitted.

This signal is emitted when the spymaster submits a hint and the number of words. It carries the hint text and the number of words as parameters.

##### Parameters

<i>hint</i>	The hint text.
<i>number</i>	The number of words associated with the hint.

#### 4.8.3.2 reset()

```
void SpymasterHint::reset ()
```

Resets the spymaster hint input fields.

This function clears the hint input field and resets the number of words to 1. It also updates the button's clickable state to ensure it is disabled until valid input is provided.

#### 4.8.3.3 submitHint

```
void SpymasterHint::submitHint () [private], [slot]
```

Slot to handle the submission of a hint.

This function retrieves the hint and number of words from the input fields, emits the hintSubmitted signal, and resets the input fields.

#### 4.8.3.4 textToUppercase

```
void SpymasterHint::textToUppercase (
    const QString & text) [private], [slot]
```

Slot to convert text to uppercase.

This function is called when the text in the hint input field changes. It converts the text to uppercase to ensure consistency in the hint format.

## Parameters

<i>text</i>	The input text.
-------------	-----------------

#### 4.8.3.5 updateButtonClickable

```
void SpymasterHint::updateButtonClickable () [private], [slot]
```

Slot to update the button's clickable state based on input.

This function checks if the hint input field is empty. If the input is valid, it enables the button; otherwise, it disables it.

### 4.8.4 Member Data Documentation

#### 4.8.4.1 giveClueButton

```
QPushButton* SpymasterHint::giveClueButton [private]
```

QPushButton to submit the hint.

#### 4.8.4.2 hintLineEdit

```
QLineEdit* SpymasterHint::hintLineEdit [private]
```

QLineEdit used by the spymaster to input the hint.

#### 4.8.4.3 numberSpinBox

```
QSpinBox* SpymasterHint::numberSpinBox [private]
```

QSpinBox used by the spymaster to input the number of words correlated to the hint.

#### 4.8.4.4 textValidator

```
QRegularExpressionValidator* SpymasterHint::textValidator [private]
```

QRegularExpressionValidator used to validate the hint the spymaster inputs is a single valid word.

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

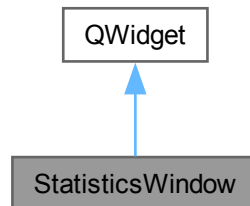
- [include/spymasterhint.h](#)
- [src/spymasterhint.cpp](#)

## 4.9 StatisticsWindow Class Reference

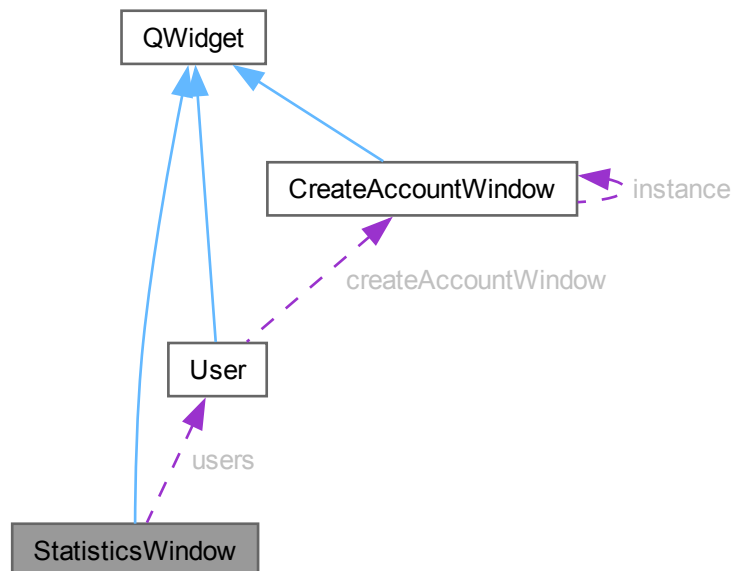
The class that shows the Statistics screen Displays game statistics for selected users including win rates and guess accuracy.

```
#include <statisticswindow.h>
```

Inheritance diagram for StatisticsWindow:



Collaboration diagram for StatisticsWindow:



### Public Slots

- void `show` ()  
*show the statistics screen Makes the statistics UI visible and updates data*



## Signals

- void [backToMainWindow](#) ()  
*Go back to the main window Signal emitted when user chooses to return to main menu.*

## Public Member Functions

- [StatisticsWindow](#) (QWidget \*parent=nullptr)  
*Construct a new Statistics Window object Initializes UI components and connects signals/slots.*
- [~StatisticsWindow](#) ()  
*Destructor for statistics screen Cleans up resources when [StatisticsWindow](#) is destroyed.*

## Private Slots

- void [goBackToMain](#) ()  
*to back to the main window Slot triggered when back button is clicked*
- void [showUserStats](#) ()  
*showing the user stats after clicking the button Slot that retrieves and displays statistics for selected user*

## Private Member Functions

- void [populateDropDown](#) ()  
*populate the drop down button with the usernames Fetches user list from [User](#) singleton and fills dropdown menu*

## Private Attributes

- [User](#) \* [users](#)  
*the users instance Singleton reference to access user data and statistics*
- QPushButton \* [backToMainButton](#)  
*button to click to go back to main UI navigation element to return to main menu*
- QComboBox \* [usernameComboBox](#)  
*the drop down box of usernames Selection widget for choosing which user's statistics to display*
- QPushButton \* [showUserStatsButton](#)  
*the button to show the user stats after choosing in drop down menu Triggers update of statistics display for selected user*
- QString [username](#)  
*the username of the user Stores the currently selected username*
- QLabel \* [usernameTitle](#)  
*title of username Display label showing selected user's name*
- QLabel \* [gamesPlayedStats](#)  
*the number of games played of the user Display label showing total games played statistic*
- QLabel \* [gamesWinStats](#)  
*the number of games win of the user Display label showing total games won statistic*
- QLabel \* [gamesWinRateStats](#)  
*the win rate of the user Display label showing win percentage (wins/games played)*
- QLabel \* [guessTotalStats](#)  
*the number of guess total of the user Display label showing total guesses made statistic*
- QLabel \* [guessHitStats](#)  
*the number of correct guess of the user Display label showing correct guesses statistic*
- QLabel \* [guessHitRateStats](#)  
*the guess hit rate of the user Display label showing guess accuracy percentage (hits/total)*

### 4.9.1 Detailed Description

The class that shows the Statistics screen Displays game statistics for selected users including win rates and guess accuracy.

### 4.9.2 Constructor & Destructor Documentation

#### 4.9.2.1 StatisticsWindow()

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

Construct a new Statistics Window object Initializes UI components and connects signals/slots.

##### Parameters

<i>parent</i>	the parent of the statistics window screen for widget hierarchy
---------------	---

#### 4.9.2.2 ~StatisticsWindow()

```
StatisticsWindow::~~StatisticsWindow ()
```

Destructor for statistics screen Cleans up resources when [StatisticsWindow](#) is destroyed.

### 4.9.3 Member Function Documentation

#### 4.9.3.1 backToMainWindow

```
void StatisticsWindow::backToMainWindow () [signal]
```

Go back to the main window Signal emitted when user chooses to return to main menu.

#### 4.9.3.2 goBackToMain

```
void StatisticsWindow::goBackToMain () [private], [slot]
```

to back to the main window Slot triggered when back button is clicked

#### 4.9.3.3 populateDropDown()

```
void StatisticsWindow::populateDropDown () [private]
```

populate the drop down button with the usernames Fetches user list from [User](#) singleton and fills dropdown menu

#### 4.9.3.4 show

```
void StatisticsWindow::show () [slot]
```

show the statistics screen Makes the statistics UI visible and updates data

#### 4.9.3.5 showUserStats

```
void StatisticsWindow::showUserStats () [private], [slot]
```

showing the user stats after clicking the button Slot that retrieves and displays statistics for selected user

### 4.9.4 Member Data Documentation

#### 4.9.4.1 backToMainButton

```
QPushButton* StatisticsWindow::backToMainButton [private]
```

button to click to go back to main UI navigation element to return to main menu

#### 4.9.4.2 gamesPlayedStats

```
QLabel* StatisticsWindow::gamesPlayedStats [private]
```

the number of games played of the user Display label showing total games played statistic

#### 4.9.4.3 gamesWinRateStats

```
QLabel* StatisticsWindow::gamesWinRateStats [private]
```

the win rate of the user Display label showing win percentage (wins/games played)

#### 4.9.4.4 gamesWinStats

```
QLabel* StatisticsWindow::gamesWinStats [private]
```

the number of games win of the user Display label showing total games won statistic

#### 4.9.4.5 guessHitRateStats

```
QLabel* StatisticsWindow::guessHitRateStats [private]
```

the guess hit rate of the user Display label showing guess accuracy percentage (hits/total)

#### 4.9.4.6 guessHitStats

```
QLabel* StatisticsWindow::guessHitStats [private]
```

the number of correct guess of the user Display label showing correct guesses statistic

#### 4.9.4.7 guessTotalStats

```
QLabel* StatisticsWindow::guessTotalStats [private]
```

the number of guess total of the user Display label showing total guesses made statistic

#### 4.9.4.8 showUserStatsButton

```
QPushButton* StatisticsWindow::showUserStatsButton [private]
```

the button to show the user stats after choosing in drop down menu Triggers update of statistics display for selected user

#### 4.9.4.9 username

```
QString StatisticsWindow::username [private]
```

the username of the user Stores the currently selected username

#### 4.9.4.10 usernameComboBox

```
QComboBox* StatisticsWindow::usernameComboBox [private]
```

the drop down box of usernames Selection widget for choosing which user's statistics to display

#### 4.9.4.11 usernameTitle

```
QLabel* StatisticsWindow::usernameTitle [private]
```

title of username Display label showing selected user's name

#### 4.9.4.12 users

```
User* StatisticsWindow::users [private]
```

the users instance Singleton reference to access user data and statistics

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

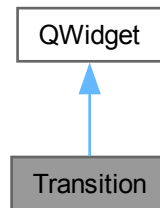
- [include/statisticswindow.h](#)
- [src/statisticswindow.cpp](#)

## 4.10 Transition Class Reference

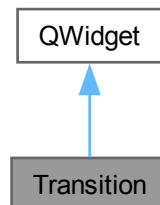
A widget for displaying a transition message and a button to continue.

```
#include <transition.h>
```

Inheritance diagram for Transition:



Collaboration diagram for Transition:



### Signals

- void `continueClicked` ()  
*Signal emitted when the continue button is clicked.*

### Public Member Functions

- `Transition` (QWidget \*parent=nullptr)  
*Constructor for the `Transition` class.*
- `~Transition` ()  
*Destructor for the `Transition` class.*
- void `setMessage` (const QString &message)  
*Sets the message to be displayed.*

### Private Attributes

- QLabel \* [messageLabel](#)  
*The label that displays the transition message.*
- QPushButton \* [continueButton](#)  
*The button that allows the user to continue.*

## 4.10.1 Detailed Description

A widget for displaying a transition message and a button to continue.

This class contains a QLabel for the message and a QPushButton to continue. It emits a signal when the button is clicked.

Author

Group 9

## 4.10.2 Constructor & Destructor Documentation

### 4.10.2.1 Transition()

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

Constructor for the [Transition](#) class.

This constructor sets up the layout and initializes the widgets. It connects the button to the continueClicked signal.

Parameters

<i>parent</i>	The parent widget.
---------------	--------------------

### 4.10.2.2 ~Transition()

```
Transition::~~Transition ()
```

Destructor for the [Transition](#) class.

This destructor cleans up the resources used by the class. It does not need to explicitly delete the widgets as they are managed by Qt's parent-child system.

## 4.10.3 Member Function Documentation

### 4.10.3.1 continueClicked

```
void Transition::continueClicked () [signal]
```

Signal emitted when the continue button is clicked.

This signal is emitted when the user clicks the continue button in the transition screen. After this signal is emitted, the game can proceed to the next state.

### 4.10.3.2 setMessage()

```
void Transition::setMessage (  
    const QString & message)
```

Sets the message to be displayed.

This function updates the text of the message label shown in the transition screen UI.

## Parameters

<i>message</i>	The message text.
----------------	-------------------

## 4.10.4 Member Data Documentation

### 4.10.4.1 `continueButton`

```
QPushButton* Transition::continueButton [private]
```

The button that allows the user to continue.

### 4.10.4.2 `messageLabel`

```
QLabel* Transition::messageLabel [private]
```

The label that displays the transition message.

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

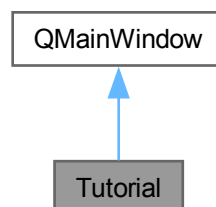
- [include/transition.h](#)
- [src/transition.cpp](#)

## 4.11 Tutorial Class Reference

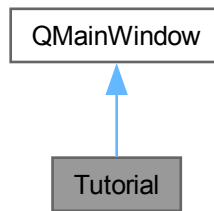
The tutorial window that guides users through the game mechanics.

```
#include <tutorial.h>
```

Inheritance diagram for Tutorial:



Collaboration diagram for Tutorial:



### Signals

- void `tutorialClosed` ()  
*Signal emitted when the tutorial is closed.*

### Public Member Functions

- `Tutorial` (QWidget \*parent=nullptr)  
*Constructor for `Tutorial`.*
- `~Tutorial` ()  
*Destructor for `Tutorial`.*

### Protected Member Functions

- void `closeEvent` (QCloseEvent \*event) override  
*Handles the close event.*

### Private Slots

- void `onContinueClicked` ()  
*Handles the continue button click event.*

### Private Member Functions

- void `updateContinueButtonPosition` ()  
*Updates the position of the continue button.*
- void `resetTutorial` ()  
*Resets the tutorial to its initial state.*



**Private Attributes**

- QWidget \* [centralWidget](#)  
*Pointer to the central widget.*
- QLabel \* [titleLabel](#)  
*Label for the tutorial title.*
- QLabel \* [textBox](#)  
*Label for displaying tutorial text.*
- QPushButton \* [continueButton](#)  
*Button for continuing through the tutorial.*
- int [clickCount](#)  
*Counter for continue button clicks.*

**4.11.1 Detailed Description**

The tutorial window that guides users through the game mechanics.

**4.11.2 Constructor & Destructor Documentation****4.11.2.1 Tutorial()**

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

Constructor for [Tutorial](#).

**Parameters**

<i>parent</i>	The parent widget (default is nullptr).
---------------	---

**4.11.2.2 ~Tutorial()**

```
Tutorial::~~Tutorial ()
```

Destructor for [Tutorial](#).

**4.11.3 Member Function Documentation****4.11.3.1 closeEvent()**

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

Handles the close event.

**Parameters**

<i>event</i>	Pointer to the close event.
--------------	-----------------------------

**4.11.3.2 onContinueClicked**

```
void Tutorial::onContinueClicked () [private], [slot]
```

Handles the continue button click event.

**4.11.3.3 resetTutorial()**

```
void Tutorial::resetTutorial () [private]
```

Resets the tutorial to its initial state.

**4.11.3.4 tutorialClosed**

```
void Tutorial::tutorialClosed () [signal]
```

Signal emitted when the tutorial is closed.

**4.11.3.5 updateContinueButtonPosition()**

```
void Tutorial::updateContinueButtonPosition () [private]
```

Updates the position of the continue button.

**4.11.4 Member Data Documentation****4.11.4.1 centralWidget**

```
QWidget* Tutorial::centralWidget [private]
```

Pointer to the central widget.

**4.11.4.2 clickCount**

```
int Tutorial::clickCount [private]
```

Counter for continue button clicks.

#### 4.11.4.3 continueButton

```
QPushButton* Tutorial::continueButton [private]
```

Button for continuing through the tutorial.

#### 4.11.4.4 textBox

```
QLabel* Tutorial::textBox [private]
```

Label for displaying tutorial text.

#### 4.11.4.5 titleLabel

```
QLabel* Tutorial::titleLabel [private]
```

Label for the tutorial title.

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

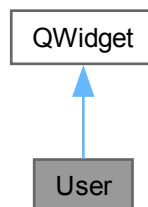
- [include/tutorial.h](#)
- [src/tutorial.cpp](#)

## 4.12 User Class Reference

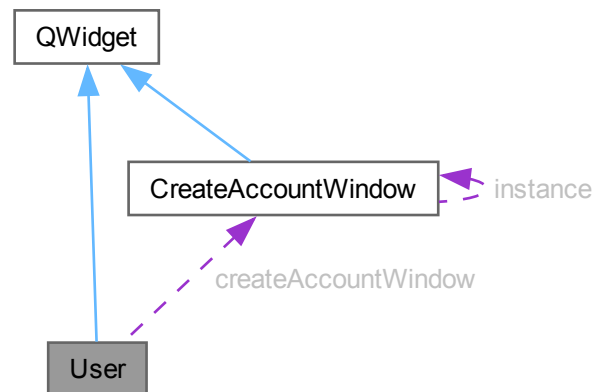
[User](#) class to handle local log in and loading/storing json files. This is a singleton class to ensure only one instance of user management exists. Manages user profiles, statistics, and authentication.

```
#include <user.h>
```

Inheritance diagram for User:



Collaboration diagram for User:



## Public Slots

- void [show](#) ()  
*show the current screen Makes the user login UI visible*

## Signals

- void [backToMainMenu](#) ()  
*signal to go to main menu Emitted when user successfully logs in or cancels login*

## Public Member Functions

- [~User](#) ()  
*Destructor of user class Cleans up resources when [User](#) object is destroyed.*
- void [updateGamesPlayed](#) (const QString &username, const unsigned int &newGamesPlayed)  
*Update the number of games played by a user Modifies user statistics and saves to profile.*
- unsigned int [getGamesPlayed](#) (const QString &username) const  
*Get the number of games played by a user Retrieves game count from user profile.*
- void [updateWins](#) (const QString &username, const unsigned int &newWins)  
*Update the number of wins a user has Modifies win statistics and saves to profile.*
- unsigned int [getWins](#) (const QString &username) const  
*Get the number of wins the user has Retrieves win count from user profile.*
- float [getWinRate](#) (const QString &username) const  
*Get the win rate of the user (games\_win/games\_played) Calculates win percentage based on games played and won.*
- void [updateGuessTotal](#) (const QString &username, const unsigned int &newGuessTotal)  
*Update the total of guesses the user has Modifies guess statistics and saves to profile.*
- unsigned int [getGuessTotal](#) (const QString &username) const  
*Get the total number of guesses the user has Retrieves guess count from user profile.*
- void [updateGuessHit](#) (const QString &username, const unsigned int &newGuessHit)

- Update the number times the user guess correctly Modifies correct guess statistics and saves to profile.*

  - unsigned int [getGuessHit](#) (const QString &username) const

*Get the number of times the user guess correctly Retrieves correct guess count from user profile.*
- float [getHitRate](#) (const QString &username)

*Get the rate the user guess correctly (guess\_hit/guess\_total) Calculates accuracy percentage based on total guesses and correct guesses.*
- void [renameUser](#) (const QString &oldUsername, const QString &newUsername)

*Rename the user Changes username in profile while preserving statistics.*
- void [won](#) (const QString &username)

*Change the games played total and games played win of the user when they won Convenience method to update multiple statistics after a win.*
- void [lost](#) (const QString &username)

*Change the games played total of the user when they lost Convenience method to update statistics after a loss.*
- void [hit](#) (const QString &username)

*Change the guess total and guess hit of the user when they guess correctly Convenience method to update multiple statistics after a correct guess.*
- void [miss](#) (const QString &username)

*Change the guess total of the user when they guess incorrectly Convenience method to update statistics after an incorrect guess.*
- QJsonObject [loadJsonFile](#) ()

*loading the info of the users Reads user profiles from JSON storage*

### Static Public Member Functions

- static [User](#) \* [instance](#) (QWidget \*parent=nullptr)

*Getting the instance of user (Singleton pattern implementation) Ensures only one instance of [User](#) class exists throughout the application.*

### Private Slots

- void [handleLogin](#) ()

*log in the user Handles authentication and session creation*
- void [refreshUserDropdown](#) ()

*refresh user info in the drop down menu Updates UI with latest user list*
- void [handleCreateAccount](#) ()

*create user account Opens account creation window*
- void [showMainMenu](#) ()

*show the main menu Returns to main application screen*

### Private Member Functions

- [User](#) (QWidget \*parent=nullptr)

*Constructor of the [User](#) instance Private to enforce singleton pattern.*
- void [populateUsernameComboBox](#) (const QJsonObject &jsonObject)

*update the usernames in the drop down when creating new users Refreshes UI with current user list*

### Private Attributes

- [CreateAccountWindow](#) \* [createAccountWindow](#)  
*variable that stores the create account window Manages account creation UI*
- QString [jsonFilePath](#) = "resources/profile.json"  
*the path of the users info Location of JSON profile storage*
- QPushButton \* [backButton](#)  
*the button to go back UI element for navigation*
- QPushButton \* [createAccountButton](#)  
*the button to create account UI element to open account creation*
- QComboBox \* [usernameComboBox](#)  
*the drop down box of the usernames of the users UI element for user selection*
- QLabel \* [jsonContentLabel](#)  
*the text to show the debug UI element for displaying information*
- QPushButton \* [loginButton](#)  
*the button to log in UI element for authentication*

## 4.12.1 Detailed Description

[User](#) class to handle local log in and loading/storing json files. This is a singleton class to ensure only one instance of user management exists. Manages user profiles, statistics, and authentication.

## 4.12.2 Constructor & Destructor Documentation

### 4.12.2.1 ~User()

```
User::~~User ()
```

Destructor of user class Cleans up resources when [User](#) object is destroyed.

### 4.12.2.2 User()

```
User::User (
    QWidget * parent = nullptr) [explicit], [private]
```

Constructor of the [User](#) instance Private to enforce singleton pattern.

#### Parameters

<i>parent</i>	the parent QWidget for memory management
---------------	--

## 4.12.3 Member Function Documentation

### 4.12.3.1 backToMainMenu

```
void User::backToMainMenu () [signal]
```

signal to go to main menu Emitted when user successfully logs in or cancels login

### 4.12.3.2 getGamesPlayed()

```
unsigned int User::getGamesPlayed (
    const QString & username) const
```

Get the number of games played by a user Retrieves game count from user profile.

**Parameters**

<i>username</i>	username of the user
-----------------	----------------------

**Returns**

`unsigned int` the number of games played

**4.12.3.3 getGuessHit()**

```
unsigned int User::getGuessHit (  
    const QString & username) const
```

Get the number of times the user guess correctly Retrieves correct guess count from user profile.

**Parameters**

<i>username</i>	username of the user
-----------------	----------------------

**Returns**

`unsigned int` the number of times the user guess correctly

**4.12.3.4 getGuessTotal()**

```
unsigned int User::getGuessTotal (  
    const QString & username) const
```

Get the total number of guesses the user has Retrieves guess count from user profile.

**Parameters**

<i>username</i>	username of the user
-----------------	----------------------

**Returns**

`unsigned int` the total number of guesses the user has

**4.12.3.5 getHitRate()**

```
float User::getHitRate (  
    const QString & username)
```

Get the rate the user guess correctly (guess\_hit/guess\_total) Calculates accuracy percentage based on total guesses and correct guesses.

**Parameters**

<i>username</i>	username of the user
-----------------	----------------------

**Returns**

float the rate the user guess correctly (guess\_hit/guess\_total)

**4.12.3.6 getWinRate()**

```
float User::getWinRate (  
    const QString & username) const
```

Get the win rate of the user (games\_win/games\_played) Calculates win percentage based on games played and won.

**Parameters**

<i>username</i>	the username of a user
-----------------	------------------------

**Returns**

float win rate of the user (games\_win/games\_played)

**4.12.3.7 getWins()**

```
unsigned int User::getWins (  
    const QString & username) const
```

Get the number of wins the user has Retrieves win count from user profile.

**Parameters**

<i>username</i>	username of the user
-----------------	----------------------

**Returns**

unsigned int the number of wins the user has

**4.12.3.8 handleCreateAccount**

```
void User::handleCreateAccount () [private], [slot]
```

create user account Opens account creation window

**4.12.3.9 handleLogin**

```
void User::handleLogin () [private], [slot]
```

log in the user Handles authentication and session creation

**4.12.3.10 hit()**

```
void User::hit (  
    const QString & username)
```

Change the guess total and guess hit of the user when they guess correctly Convenience method to update multiple statistics after a correct guess.



## Parameters

<i>username</i>	username of the user
-----------------	----------------------

**4.12.3.11 instance()**

```
User * User::instance (
    QWidget * parent = nullptr) [static]
```

Getting the instance of user (Singleton pattern implementation) Ensures only one instance of [User](#) class exists throughout the application.

## Parameters

<i>parent</i>	parent QWidget for ownership hierarchy
---------------	--

## Returns

\*[User](#) Pointer to the single [User](#) instance

**4.12.3.12 loadJsonFile()**

```
QJsonObject User::loadJsonFile ()
```

loading the info of the users Reads user profiles from JSON storage

## Returns

QJsonObject the info of the user in json format

**4.12.3.13 lost()**

```
void User::lost (
    const QString & username)
```

Change the games played total of the user when they lost Convenience method to update statistics after a loss.

## Parameters

<i>username</i>	username of the user
-----------------	----------------------

**4.12.3.14 miss()**

```
void User::miss (
    const QString & username)
```

Change the guess total of the user when they guess incorrectly Convenience method to update statistics after an incorrect guess.

**Parameters**

<i>username</i>	username of the user
-----------------	----------------------

**4.12.3.15 populateUsernameComboBox()**

```
void User::populateUsernameComboBox (  
    const QJsonObject & jsonObject) [private]
```

update the usernames in the drop down when creating new users Refreshes UI with current user list

**Parameters**

<i>jsonObject</i>	the json of the users
-------------------	-----------------------

**4.12.3.16 refreshUserDropdown**

```
void User::refreshUserDropdown () [private], [slot]
```

refresh user info in the drop down menu Updates UI with latest user list

**4.12.3.17 renameUser()**

```
void User::renameUser (  
    const QString & oldUsername,  
    const QString & newUsername)
```

Rename the user Changes username in profile while preserving statistics.

**Parameters**

<i>oldUsername</i>	old username of the user
<i>newUsername</i>	new username of the user

**4.12.3.18 show**

```
void User::show () [slot]
```

show the current screen Makes the user login UI visible

**4.12.3.19 showMainMenu**

```
void User::showMainMenu () [private], [slot]
```

show the main menu Returns to main application screen

**4.12.3.20 updateGamesPlayed()**

```
void User::updateGamesPlayed (  
    const QString & username,  
    const unsigned int & newGamesPlayed)
```

Update the number of games played by a user Modifies user statistics and saves to profile.

## Parameters

<i>username</i>	username of the user to update
<i>newGamesPlayed</i>	the new number of games played by a user

**4.12.3.21 updateGuessHit()**

```
void User::updateGuessHit (  
    const QString & username,  
    const unsigned int & newGuessHit)
```

Update the number times the user guess correctly Modifies correct guess statistics and saves to profile.

## Parameters

<i>username</i>	username of the user
<i>newGuessHit</i>	the number of times the user guess correctly

**4.12.3.22 updateGuessTotal()**

```
void User::updateGuessTotal (  
    const QString & username,  
    const unsigned int & newGuessTotal)
```

Update the total of guesses the user has Modifies guess statistics and saves to profile.

## Parameters

<i>username</i>	username of the user
<i>newGuessTotal</i>	the new total number of guesses the user has

**4.12.3.23 updateWins()**

```
void User::updateWins (  
    const QString & username,  
    const unsigned int & newWins)
```

Update the number of wins a user has Modifies win statistics and saves to profile.

## Parameters

<i>username</i>	username of the user
<i>newWins</i>	the new number of wins the user has

**4.12.3.24 won()**

```
void User::won (  
    const QString & username)
```

Change the games played total and games played win of the user when they won Convenience method to update multiple statistics after a win.

## Parameters

<code>username</code>	username of the user
-----------------------	----------------------

## 4.12.4 Member Data Documentation

### 4.12.4.1 backButton

```
QPushButton* User::backButton [private]
```

the button to go back UI element for navigation

### 4.12.4.2 createAccountButton

```
QPushButton* User::createAccountButton [private]
```

the button to create account UI element to open account creation

### 4.12.4.3 createAccountWindow

```
CreateAccountWindow* User::createAccountWindow [private]
```

variable that stores the create account window Manages account creation UI

### 4.12.4.4 jsonContentLabel

```
QLabel* User::jsonContentLabel [private]
```

the text to show the debug UI element for displaying information

### 4.12.4.5 jsonFilePath

```
QString User::jsonFilePath = "resources/profile.json" [private]
```

the path of the users info Location of JSON profile storage

### 4.12.4.6 loginButton

```
QPushButton* User::loginButton [private]
```

the button to log in UI element for authentication

### 4.12.4.7 usernameComboBox

```
QComboBox* User::usernameComboBox [private]
```

the drop down box of the usernames of the users UI element for user selection

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

- [include/user.h](#)
- [src/user.cpp](#)

## Chapter 5

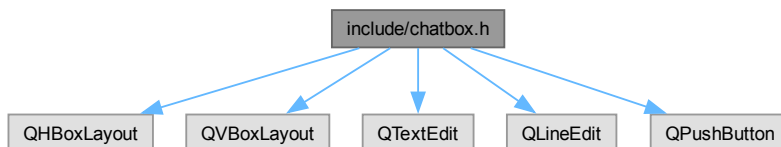
# File Documentation

### 5.1 include/chatbox.h File Reference

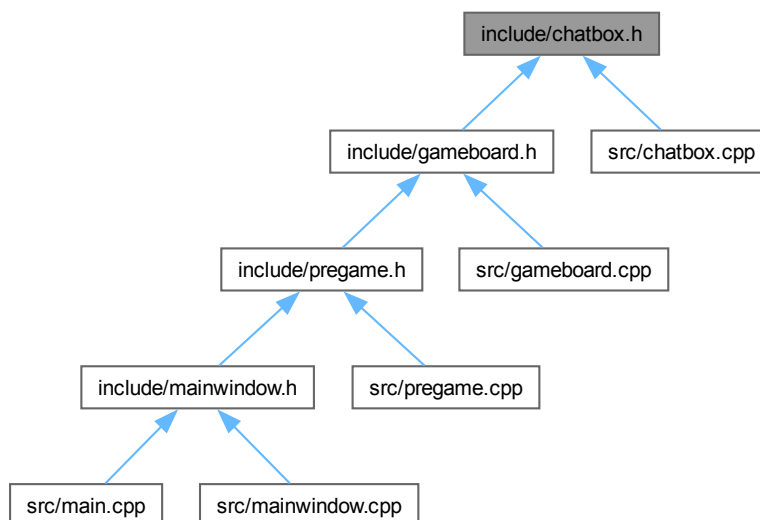
Header file for the [ChatBox](#) class, which provides a UI for the chat feature in the game.

```
#include <QHBoxLayout>
#include <QVBoxLayout>
#include <QTextEdit>
#include <QLineEdit>
#include <QPushButton>
```

Include dependency graph for chatbox.h:



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



## Classes

- class [ChatBox](#)

*A widget for the chat feature in the game.*

### 5.1.1 Detailed Description

Header file for the [ChatBox](#) class, which provides a UI for the chat feature in the game.

#### Author

Matthew Marbina (Group 9)

#### Version

0.1

#### Date

2025-03-30

#### Copyright

Copyright (c) 2025

## 5.2 chatbox.h

[Go to the documentation of this file.](#)

```

00001
00010 #ifndef CHATBOX_H
00011 #define CHATBOX_H
00012
00013 #include <QHBoxLayout>
00014 #include <QVBoxLayout>
00015 #include <QTextEdit>
00016 #include <QLineEdit>
00017 #include <QPushButton>
00018
00028 class ChatBox : public QWidget {
00029     Q_OBJECT
00030
00031 public:
00038     enum Team {
00039         RED_TEAM,
00040         BLUE_TEAM
00041     };
00042
00051     explicit ChatBox(const QString& playerName, Team team, QWidget* parent = nullptr);
00052
00058     ~ChatBox();
00059
00067     void addSystemMessage(const QString& message, Team team);
00068
00075     void addPlayerMessage(const QString& playerName, const QString& message);
00076
00082     void setPlayerName(const QString& name);
00083
00088     void clearChat();
00089
00095     void limitReachedMessage();
00096
00097 public slots:
00103     void sendMessage();
00104 signals:
00112     void massSend(const QString& playerName, const QString& message);
00113
00114 private:
00118     Team team;
00119
00123     QTextEdit* chatDisplay;
00124
00128     QLineEdit* chatInput;
00129
00133     QPushButton* sendButton;
00134
00138     QString playerName;
00139 };
00140
00141 #endif // CHATBOX_H

```

## 5.3 include/createaccountwindow.h File Reference

Header file for the [CreateAccountWindow](#) class which handles user account creation.

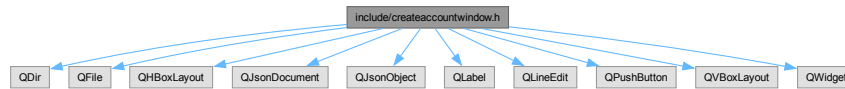
```

#include <QDir>
#include <QFile>
#include <QHBoxLayout>
#include <QJsonDocument>
#include <QJsonObject>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QVBoxLayout>

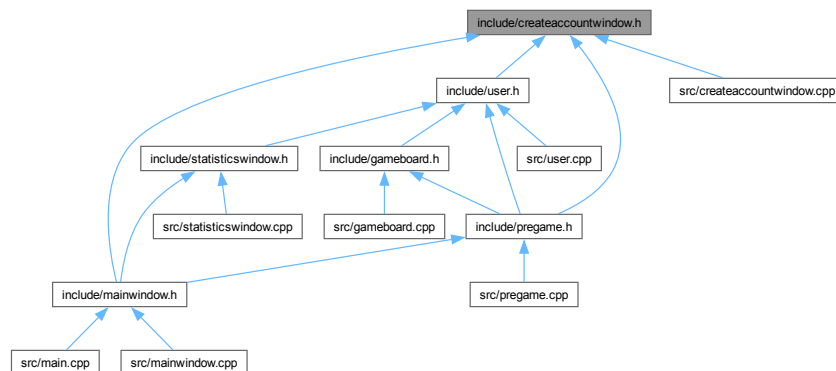
```

```
#include <QWidget>
```

Include dependency graph for createaccountwindow.h:



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



## Classes

- class [CreateAccountWindow](#)

The [CreateAccountWindow](#) class provides a singleton interface for creating new user accounts. This window allows users to input a username and creates a profile JSON file for the new account.

### 5.3.1 Detailed Description

Header file for the [CreateAccountWindow](#) class which handles user account creation.

#### Author

Team 9 - UWO CS 3307

#### Version

0.1

#### Date

2025-03-30

#### Copyright

Copyright (c) 2025



## 5.4 createaccountwindow.h

[Go to the documentation of this file.](#)

```

00001
00012
00013 #ifndef CREATEACCOUNTWINDOW_H
00014 #define CREATEACCOUNTWINDOW_H
00015
00016 #include <QDir>
00017 #include <QFile>
00018 #include <QHBoxLayout>
00019 #include <QJsonDocument>
00020 #include <QJsonObject>
00021 #include <QLabel>
00022 #include <QLineEdit>
00023 #include <QPushButton>
00024 #include <QVBoxLayout>
00025 #include <QWidget>
00026
00032 class CreateAccountWindow : public QWidget {
00033     Q_OBJECT
00034
00035 public:
00043     static CreateAccountWindow* getInstance(QWidget* parent = nullptr);
00044
00051     void setPreviousScreen(QWidget* previous);
00052
00053 public slots:
00058     void show();
00059
00060 private:
00067     explicit CreateAccountWindow(QWidget* parent = nullptr);
00068
00073     static CreateAccountWindow* instance;
00074
00075 private slots:
00080     void onCreateAccountClicked();
00081
00086     void goBack();
00087
00088 signals:
00093     void back();
00094
00099     void accountCreated();
00100
00101 private:
00108     void saveJsonFile(const QString& username);
00109
00113     QLineEdit* usernameEdit;
00114
00118     QPushButton* createAccountButton;
00119
00123     QLabel* statusLabel;
00124
00129     QString jsonFilePath = "resources/profile.json"; // Update path as necessary
00130
00135     QWidget* previousScreen = nullptr;
00136 };
00137
00138 #endif // CREATEACCOUNTWINDOW_H

```

## 5.5 include/gameboard.h File Reference

Header file for the [GameBoard](#) class, which implements a game board for the Spy Master game.

```

#include <QDebug>
#include <QFile>
#include <QGridLayout>
#include <QLabel>
#include <QMessageBox>
#include <QPushButton>
#include <QRandomGenerator>
#include <QStackedLayout>

```



## 5.6 gameboard.h

[Go to the documentation of this file.](#)

```

00001
00007
00008 #ifndef GAMEBOARD_H
00009 #define GAMEBOARD_H
00010
00011 #include <QDebug>
00012 #include <QFile>
00013 #include <QGridLayout>
00014 #include <QLabel>
00015 #include <QMessageBox>
00016 #include <QPushButton>
00017 #include <QRandomGenerator>
00018 #include <QStackedLayout>
00019 #include <QStringList>
00020 #include <QTextStream>
00021 #include <QVBoxLayout>
00022 #include <QWidget>
00023
00024 #include "chatbox.h"
00025 #include "operatororguess.h"
00026 #include "spymasterhint.h"
00027 #include "transition.h"
00028 #include "user.h"
00029
00044
00045 class GameBoard : public QWidget {
00046     Q_OBJECT
00047
00048 public:
00064     explicit GameBoard(const QString& redSpyMaster, const QString& redOperative,
00065                       const QString& blueSpyMaster, const QString& blueOperative,
00066                       QWidget* parent = nullptr);
00067
00075     ~GameBoard();
00076
00086     void setRedSpyMasterName(const QString& name);
00087
00098     void setRedOperativeName(const QString& name);
00099
00110     void setBlueSpyMasterName(const QString& name);
00111
00122     void setBlueOperativeName(const QString& name);
00123
00132     void updateTeamLabels();
00133
00134 signals:
00143     void gameEnded();
00144
00145 public slots:
00153     void show();
00154
00166     void displayHint(const QString& hint, int number);
00167
00176     void displayGuess();
00177
00178 private:
00186     void loadWordsFromFile();
00187
00195     void generateGameGrid();
00196
00205     void setupUI();
00206
00214     void nextTurn();
00215
00226     void onCardClicked(int row, int col);
00227
00235     void onContinueClicked();
00236
00244     void showTransition();
00245
00254     void updateScores();
00255
00264     void checkGameEnd();
00265
00275     void endGame(const QString& message);
00276
00284     void resetGame();
00285
00290     enum CardType { RED_TEAM, BLUE_TEAM, NEUTRAL, ASSASSIN };
00291
00297     enum Turn { RED_SPY, RED_OP, BLUE_SPY, BLUE_OP };
00298

```

```

00304     struct Card {
00305         QString word;
00306         CardType type;
00307         bool revealed;
00308     };
00309
00311     int currentTurn;
00313     int redCardsRemaining;
00315     int blueCardsRemaining;
00316
00318     int maxGuesses = 0;
00320     int currentGuesses = 0;
00321
00323     QString redSpyMasterName;
00325     QString redOperativeName;
00327     QString blueSpyMasterName;
00329     QString blueOperativeName;
00330
00332     static const int GRID_SIZE = 5;
00334     Card gameGrid[GRID_SIZE][GRID_SIZE];
00336     QStringList wordList;
00337
00339     QGridLayout* gridLayout;
00341     QPushButton* cards[GRID_SIZE][GRID_SIZE];
00342
00344     QLabel* redTeamLabel;
00346     QLabel* blueTeamLabel;
00348     QLabel* currentTurnLabel;
00349
00351     SpymasterHint* spymasterHint;
00353     OperatorGuess* operatorGuess;
00355     QLabel* currentHint;
00357     QString correspondingNumber;
00358
00360     Transition* transition;
00361
00363     QLabel* redScoreLabel;
00365     QLabel* blueScoreLabel;
00366
00368     ChatBox* chatBox;
00370     QString currentPlayerName;
00372     ChatBox::Team currentPlayerTeam;
00374     User* users;
00375 };
00376
00377 #endif // GAMEBOARD_H

```

## 5.7 include/mainwindow.h File Reference

Declaration of the [MainWindow](#) class.

```

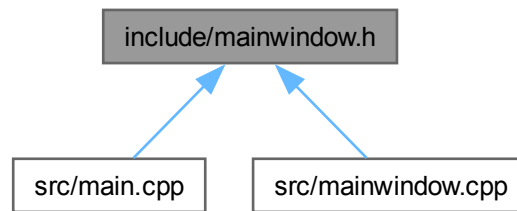
#include <QGraphicsDropShadowEffect>
#include <QGuiApplication>
#include <QLabel>
#include <QMainWindow>
#include <QPalette>
#include <QPixmap>
#include <QPushButton>
#include <QScreen>
#include <QVBoxLayout>
#include "createaccountwindow.h"
#include "pregame.h"
#include "statisticswindow.h"
#include "tutorial.h"

```

Include dependency graph for mainwindow.h:



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



## Classes

- class [MainWindow](#)  
*The main application window.*

### 5.7.1 Detailed Description

Declaration of the [MainWindow](#) class.

## 5.8 mainwindow.h

[Go to the documentation of this file.](#)

```

00001
00005
00006 #ifndef MAINWINDOW_H
00007 #define MAINWINDOW_H
00008
00009 #include <QGraphicsDropShadowEffect>
00010 #include <QGuiApplication>
00011 #include <QLabel>
00012 #include <QMainWindow>
00013 #include <QPalette>
00014 #include <QPixmap>
00015 #include <QPushButton>
00016 #include <QScreen>
00017 #include <QVBoxLayout>
00018
00019 #include "createaccountwindow.h"
00020 #include "pregame.h"
00021 #include "statisticswindow.h"
00022 #include "tutorial.h"
00023
00024 class PreGame;
00025 class CreateAccountWindow;
00026 class StatisticsWindow;
00027 class Tutorial;
00028
00033 class MainWindow : public QMainWindow {
00034     Q_OBJECT
00035
00036 public:
00041     explicit MainWindow(QWidget* parent = nullptr);
00042
00046     ~MainWindow();
00047
00048 public slots:
00052     void showMainWindow();
  
```

```

00053
00054 private slots:
00055     void openPreGame();
00056
00057     void openStatsWindow();
00058
00059     void openCreateAccount();
00060
00061     void openTutorial();
00062
00063     void openMultiMain();
00064
00065 private:
00066     QWidget* centralWidget;
00067     QVBoxLayout* layout;
00068
00069     QLabel* titleLabel;
00070
00071     PreGame* preGameWindow;
00072     QPushButton* localPlayButton;
00073     QPushButton* onlinePlayButton;
00074     QPushButton* tutorialButton;
00075     QPushButton* statsButton;
00076     QPushButton*
00077         createAccountButton;
00078
00079     CreateAccountWindow*
00080         createAccountWindow;
00081     StatisticsWindow*
00082         statsWindow;
00083     Tutorial* tutorialWindow;
00084 };
00085
00086 #endif // MAINWINDOW_H

```

## 5.9 include/operatorguess.h File Reference

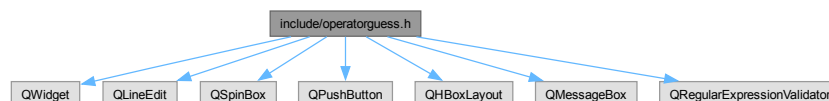
Header file for the [OperatorGuess](#) class, which handles operator guessing interface.

```

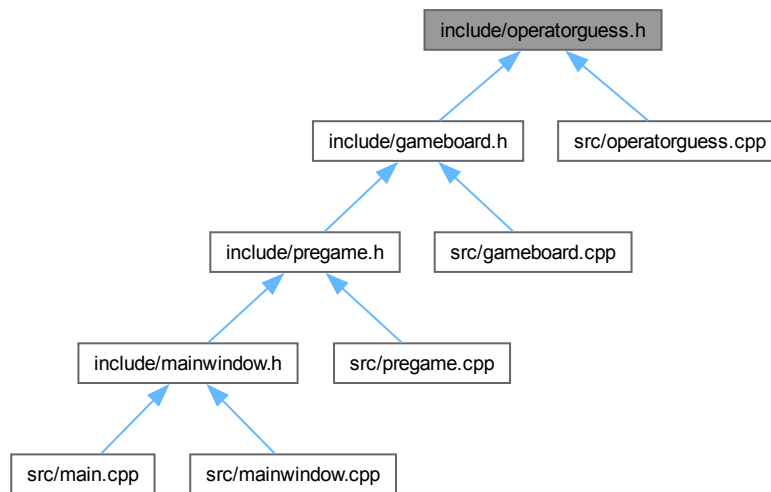
#include <QWidget>
#include <QLineEdit>
#include <QSpinBox>
#include <QPushButton>
#include <QHBoxLayout>
#include <QMessageBox>
#include <QRegularExpressionValidator>

```

Include dependency graph for operatorguess.h:



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



## Classes

- class [OperatorGuess](#)

*A widget that provides the interface for operators to submit guesses during gameplay.*

## 5.9.1 Detailed Description

Header file for the [OperatorGuess](#) class, which handles operator guessing interface.

Author

Group 9

## 5.10 operatorguess.h

[Go to the documentation of this file.](#)

```

00001
00006
00007 #ifndef OPERATORGUESS_H
00008 #define OPERATORGUESS_H
00009
00010 #include <QWidget>
00011 #include <QLineEdit>
00012 #include <QSpinBox>
00013 #include <QPushButton>
00014 #include <QHBoxLayout>
00015 #include <QMessageBox>
00016 #include <QRegularExpressionValidator>
00017
00028 class OperatorGuess : public QWidget {
00029     Q_OBJECT
00030
00031 public:
00042     explicit OperatorGuess(QWidget* parent = nullptr);

```

```

00043
00051     ~OperatorGuess();
00052
00061     void reset();
00062
00063 signals:
00070     void guessSubmitted();
00071
00072 private slots:
00081     void submitGuess();
00082
00083 private:
00084
00086     QPushButton* submitGuessButton;
00087 };
00088
00089 #endif // OPERATORGUESS_H

```

## 5.11 include/pregame.h File Reference

Header file for the [PreGame](#) class which handles the game setup screen.

```

#include <QComboBox>
#include <QDebug>
#include <QGuiApplication>
#include <QHBoxLayout>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QScreen>
#include <QVBoxLayout>
#include <QWidget>
#include "createaccountwindow.h"
#include "gameboard.h"
#include "user.h"

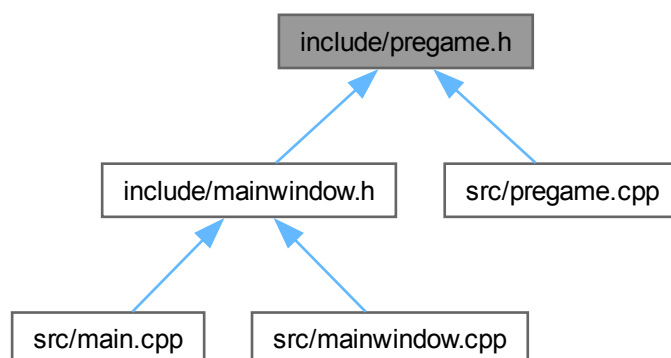
```

Include dependency graph for pregame.h:





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



## Classes

- class [PreGame](#)

*The [PreGame](#) class provides the interface for setting up a new game This includes selecting players for each team and role before starting the game.*

### 5.11.1 Detailed Description

Header file for the [PreGame](#) class which handles the game setup screen.

#### Author

Team 9 - UWO CS 3307

#### Version

0.1

#### Date

2025-03-30

#### Copyright

Copyright (c) 2025

## 5.12 pregame.h

[Go to the documentation of this file.](#)

```

00001
00011
00012 #ifndef PREGAME_H
00013 #define PREGAME_H
00014
00015 #include <QComboBox>
00016 #include <QDebug>
00017 #include <QGuiApplication>
00018 #include <QHBoxLayout>
00019 #include <QLabel>
00020 #include <QLineEdit>
00021 #include <QPushButton>
00022 #include <QScreen>
00023 #include <QVBoxLayout>
00024 #include <QWidget>
00025
00026 #include "createaccountwindow.h"
00027 #include "gameboard.h"
00028 #include "user.h"
00029
00030 class User;
00031 class CreateAccountWindow;
00032
00033 class PreGame : public QWidget {
00034     Q_OBJECT
00040
00041 public:
00047     explicit PreGame(QWidget* parent = nullptr);
00048
00053     ~PreGame();
00054
00060     QString getRedTeamSpyMasterNickname() const;
00061
00067     QString getRedTeamOperativeNickname() const;
00068
00074     QString getBlueTeamSpyMasterNickname() const;
00075
00081     QString getBlueTeamOperativeNickname() const;
00082
00083 public slots:
00088     void show();
00089
00090 private:
00097     void populateUserDropdowns();
00098
00099 private slots:
00105     void goBackToMain();
00106
00112     void startGame();
00113
00119     void handleGameEnd();
00120
00126     void openCreateAccount();
00127
00128 signals:
00134     void backToMainWindow();
00135
00141     void start();
00142
00148     void update();
00149
00150 private:
00156     User* users;
00157
00163     QStringList usernames;
00164
00170     CreateAccountWindow* createAccountWindow;
00171
00176     QLabel* label;
00177
00182     QPushButton* backButton;
00183
00188     QPushButton* createAccountButton;
00189
00194     QPushButton* startButton;
00195
00200     QComboBox* redTeamSpyMasterComboBox;
00201
00206     QComboBox* redTeamOperativeComboBox;
00207
00212     QComboBox* blueTeamSpyMasterComboBox;
00213

```

```

00218   QComboBox* blueTeamOperativeComboBox;
00219
00224   QVBoxLayout* layout;
00225
00230   QHBoxLayout* teamsLayout;
00231
00236   QVBoxLayout* redTeamLayout;
00237
00242   QVBoxLayout* blueTeamLayout;
00243
00248   QHBoxLayout* buttonsLayout;
00249
00254   GameBoard* gameBoard;
00255 };
00256
00257 #endif // PREGAME_H

```

## 5.13 include/spymasterhint.h File Reference

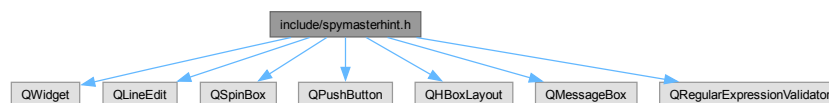
Header file for the [SpymasterHint](#) class, which provides a UI for the spymaster to give hints.

```

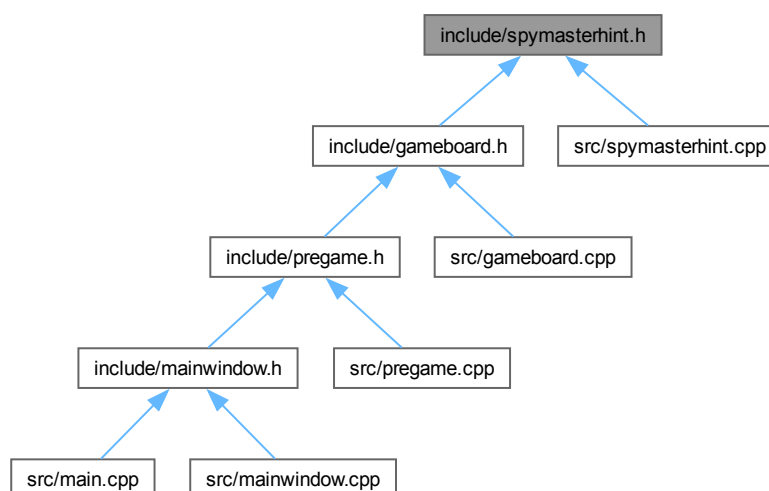
#include <QWidget>
#include <QLineEdit>
#include <QSpinBox>
#include <QPushButton>
#include <QHBoxLayout>
#include <QMessageBox>
#include <QRegularExpressionValidator>

```

Include dependency graph for spymasterhint.h:



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



## Classes

- class [SpymasterHint](#)

*A widget for the spymaster to input a hint and the number of words associated with it.*

### 5.13.1 Detailed Description

Header file for the [SpymasterHint](#) class, which provides a UI for the spymaster to give hints.

#### Author

Matthew Marbina (Group 9)

#### Version

0.1

#### Date

2025-03-30

#### Copyright

Copyright (c) 2025

## 5.14 spymasterhint.h

[Go to the documentation of this file.](#)

```

00001
00010 #ifndef SPYMASTERHINT_H
00011 #define SPYMASTERHINT_H
00012
00013 #include <QWidget>
00014 #include <QLineEdit>
00015 #include <QSpinBox>
00016 #include <QPushButton>
00017 #include <QHBoxLayout>
00018 #include <QMessageBox>
00019 #include <QRegularExpressionValidator>
00020
00029 class SpymasterHint : public QWidget {
00030     Q_OBJECT
00031
00032 public:
00041     explicit SpymasterHint(QWidget* parent = nullptr);
00042
00048     ~SpymasterHint();
00049
00055     void reset();
00056
00057 signals:
00065     void hintSubmitted(const QString& hint, const int number);
00066
00067 private slots:
00073     void submitHint();
00074
00080     void updateButtonClickable();
00081
00088     void textToUppercase(const QString& text);
00089
00090 private:
00094     QLineEdit* hintLineEdit;
00095
00099     QSpinBox* numberSpinBox;
00100
00104     QPushButton* giveClueButton;
00105
00109     QRegularExpressionValidator* textValidator;
00110 };
00111
00112 #endif // SPYMASTERHINT_H

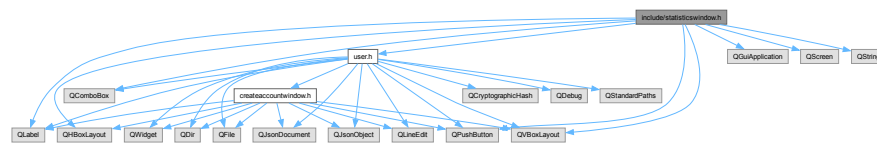
```

## 5.15 include/statisticswindow.h File Reference

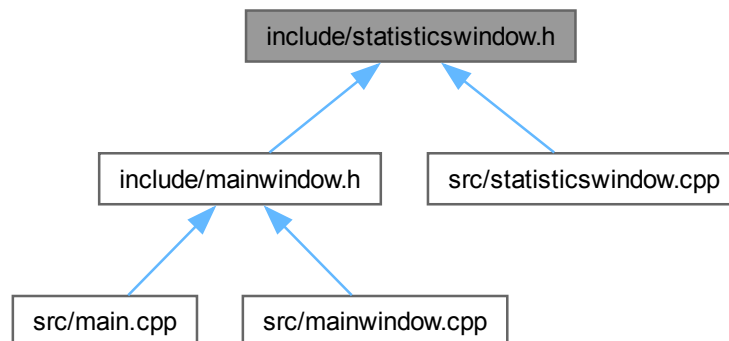
The screen to show the user's statistics.

```
#include <QComboBox>
#include <QGuiApplication>
#include <QHBoxLayout>
#include <QLabel>
#include <QPushButton>
#include <QScreen>
#include <QString>
#include <QVBoxLayout>
#include "user.h"
```

Include dependency graph for statisticswindow.h:



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



## Classes

- class **StatisticsWindow**

*The class that shows the Statistics screen Displays game statistics for selected users including win rates and guess accuracy.*

### 5.15.1 Detailed Description

The screen to show the user's statistics.

**Author**

Team 9 - UWO CS 3307

**Version**

0.1

**Date**

2025-03-30

**Copyright**

Copyright (c) 2025

## 5.16 statisticswindow.h

[Go to the documentation of this file.](#)

```
00001
00011 #ifndef STATISTICS_WINDOW_H
00012 #define STATISTICS_WINDOW_H
00013
00014 // Qt framework includes for UI components and screen management
00015 #include <QComboBox> // For dropdown menu of usernames
00016 #include <QGuiApplication> // For application-level GUI functionality
00017 #include <QHBoxLayout> // For horizontal layout arrangement
00018 #include <QLabel> // For text display in UI
00019 #include <QPushButton> // For button UI elements
00020 #include <QScreen> // For screen geometry information
00021 #include <QString> // For string handling
00022 #include <QVBoxLayout> // For vertical layout arrangement
00023
00024 #include "user.h" // Include for user data access
00025
00026 // Forward declaration to resolve circular dependency
00027 class User;
00028
00034 class StatisticsWindow : public QWidget {
00035     Q_OBJECT // Qt macro for enabling signals and slots mechanism
00036
00037     signals :
00042     void
00043     backToMainWindow();
00044
00045 public:
00053     explicit StatisticsWindow(QWidget* parent = nullptr);
00054
00059     ~StatisticsWindow();
00060
00061 public slots:
00066     void show();
00067
00068 private:
00073     User* users;
00074
00079     QPushButton* backToMainButton;
00080
00085     QComboBox* usernameComboBox;
00086
00091     QPushButton* showUserStatsButton;
00092
00097     QString username;
00098
00103     QLabel* usernameTitle;
00104
00109     QLabel* gamesPlayedStats;
00110
00115     QLabel* gamesWinStats;
00116
00121     QLabel* gamesWinRateStats;
00122
```

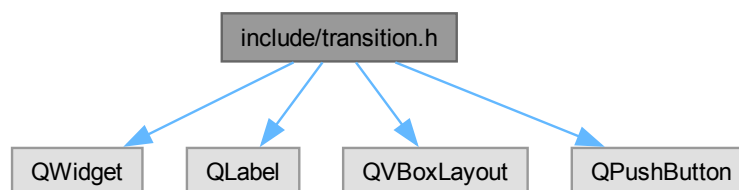
```
00127   QLabel* guessTotalStats;
00128
00133   QLabel* guessHitStats;
00134
00139   QLabel* guessHitRateStats;
00140
00141 private:
00146   void populateDropDown();
00147
00148 private slots:
00153   void goBackToMain();
00154
00159   void showUserStats();
00160 };
00161
00162 #endif // STATISTICS_WINDOW_H
```

## 5.17 include/transition.h File Reference

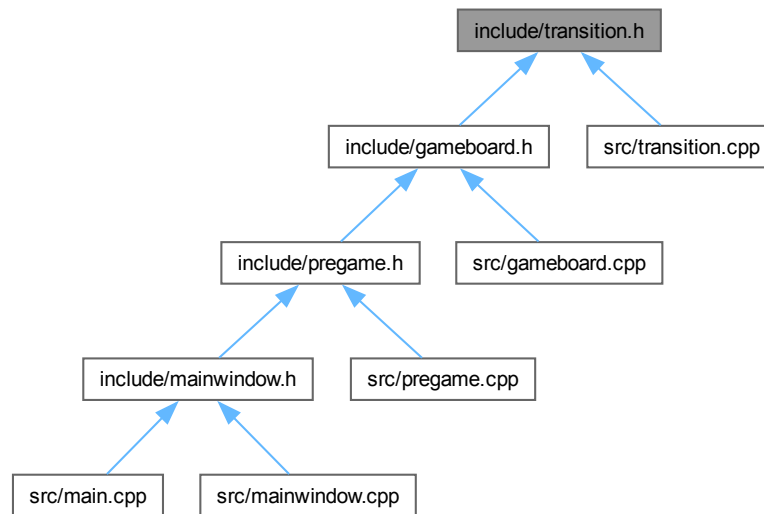
Header file for the [Transition](#) class, which provides a UI for transitions between game states.

```
#include <QWidget>
#include <QLabel>
#include <QVBoxLayout>
#include <QPushButton>
```

Include dependency graph for transition.h:



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



## Classes

- class [Transition](#)

*A widget for displaying a transition message and a button to continue.*

### 5.17.1 Detailed Description

Header file for the [Transition](#) class, which provides a UI for transitions between game states.

#### Author

Matthew Marbina (Group 9)

#### Version

0.1

#### Date

2025-03-30

#### Copyright

Copyright (c) 2025



## 5.18 transition.h

[Go to the documentation of this file.](#)

```

00001
00010 #ifndef TRANSITION_H
00011 #define TRANSITION_H
00012
00013 #include <QWidget>
00014 #include <QLabel>
00015 #include <QVBoxLayout>
00016 #include <QPushButton>
00017
00025 class Transition : public QWidget {
00026     Q_OBJECT
00027
00028 public:
00035     explicit Transition(QWidget* parent = nullptr);
00036
00042     ~Transition();
00043
00049     void setMessage(const QString& message);
00050
00051 signals:
00057     void continueClicked();
00058
00059 private:
00063     QLabel* messageLabel;
00064
00068     QPushButton* continueButton;
00069 };
00070
00071 #endif // TRANSITION_H

```

## 5.19 include/tutorial.h File Reference

Declaration of the [Tutorial](#) class.

```

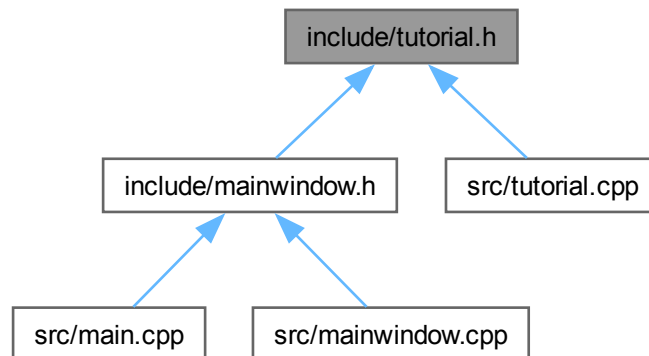
#include <QCloseEvent>
#include <QDebug>
#include <QDir>
#include <QFile>
#include <QGraphicsDropShadowEffect>
#include <QGuiApplication>
#include <QLabel>
#include <QMainWindow>
#include <QPushButton>
#include <QScreen>
#include <QVBoxLayout>
#include <QWidget>

```

Include dependency graph for tutorial.h:



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



## Classes

- class [Tutorial](#)

*The tutorial window that guides users through the game mechanics.*

### 5.19.1 Detailed Description

Declaration of the [Tutorial](#) class.

## 5.20 tutorial.h

[Go to the documentation of this file.](#)

```

00001
00005
00006 #ifndef TUTORIAL_H
00007 #define TUTORIAL_H
00008
00009 #include <QCloseEvent>
00010 #include <QDebug>
00011 #include <QDir>
00012 #include <QFile>
00013 #include <QGraphicsDropShadowEffect>
00014 #include <QGuiApplication>
00015 #include <QLabel>
00016 #include <QMainWindow>
00017 #include <QPushButton>
00018 #include <QScreen>
00019 #include <QVBoxLayout>
00020 #include <QWidget>
00021
00026 class Tutorial : public QMainWindow {
00027     Q_OBJECT
00028
00029 public:
00034     explicit Tutorial(QWidget* parent = nullptr);
00035
00039     ~Tutorial();
00040
00041 signals:
00045     void tutorialClosed();
  
```

```

00046
00047 protected:
00052 void closeEvent(QCloseEvent* event) override;
00053
00054 private slots:
00058 void onContinueClicked();
00059
00060 private:
00064 void updateContinueButtonPosition();
00065
00069 void resetTutorial();
00070
00071 QWidget* centralWidget;
00072 QLabel* titleLabel;
00073 QLabel* textBox;
00074 QPushButton* continueButton;
00075 int clickCount;
00076 };
00077
00078 #endif // TUTORIAL_H

```

## 5.21 include/user.h File Reference

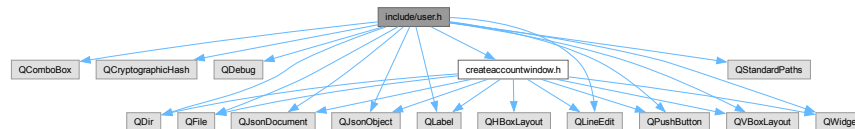
[User](#) class to handle local log in and loading/storing json files.

```

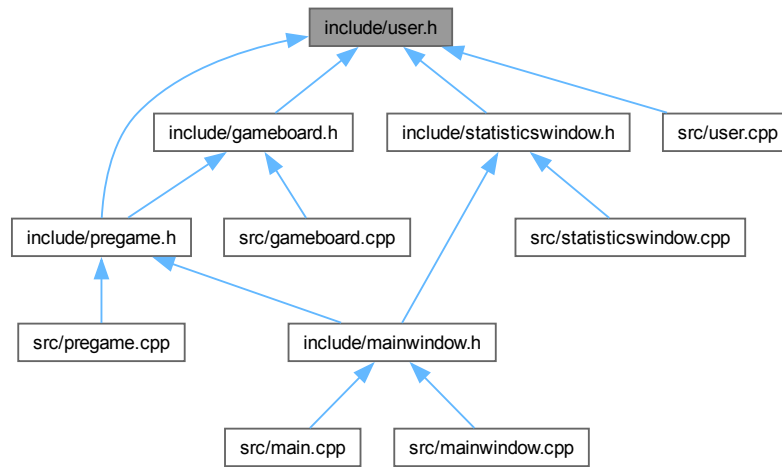
#include <QComboBox>
#include <QCryptographicHash>
#include <QDebug>
#include <QDir>
#include <QFile>
#include <QJsonDocument>
#include <QJsonObject>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QStandardPaths>
#include <QVBoxLayout>
#include <QWidget>
#include "createaccountwindow.h"

```

Include dependency graph for user.h:



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



## Classes

- class [User](#)

[User](#) class to handle local log in and loading/storing json files. This is a singleton class to ensure only one instance of user management exists. Manages user profiles, statistics, and authentication.

### 5.21.1 Detailed Description

[User](#) class to handle local log in and loading/storing json files.

#### Author

Team 9 - UWO CS 3307

#### Version

0.1

#### Date

2025-03-30

#### Copyright

Copyright (c) 2025

## 5.22 user.h

[Go to the documentation of this file.](#)

```

00001
00011 #ifndef USER_H
00012 #define USER_H
00013
00014 // Qt framework includes for UI components and file handling
00015 #include <QComboBox> // For dropdown menu of usernames
00016 #include <QCryptographicHash> // For password hashing functionality
00017 #include <QDebug> // For debug output to console
00018 #include <QDir> // For directory manipulation
00019 #include <QFile> // For file I/O operations
00020 #include <QJsonDocument> // For JSON document parsing
00021 #include <QJsonObject> // For JSON object manipulation
00022 #include <QLabel> // For text display in UI
00023 #include <QLineEdit> // For text input fields
00024 #include <QPushButton> // For button UI elements
00025 #include <QStandardPaths> // For accessing standard file locations
00026 #include <QVBoxLayout> // For vertical layout arrangement
00027 #include <QWidget> // Base class for all UI elements
00028
00029 #include "createaccountwindow.h" // Include for account creation UI
00030
00031 // Forward declaration to resolve circular dependency
00032 class CreateAccountWindow;
00033
00034 class User : public QWidget {
00035     Q_OBJECT // Qt macro for enabling signals and slots mechanism
00036
00037     public :
00038         static User*
00039         instance(QWidget* parent = nullptr);
00040
00041     ~User();
00042
00043     void updateGamesPlayed(const QString& username,
00044                             const unsigned int& newGamesPlayed);
00045
00046     unsigned int getGamesPlayed(const QString& username) const;
00047
00048     void updateWins(const QString& username, const unsigned int& newWins);
00049
00050     unsigned int getWins(const QString& username) const;
00051
00052     float getWinRate(const QString& username) const;
00053
00054     void updateGuessTotal(const QString& username,
00055                             const unsigned int& newGuessTotal);
00056
00057     unsigned int getGuessTotal(const QString& username) const;
00058
00059     void updateGuessHit(const QString& username, const unsigned int& newGuessHit);
00060
00061     unsigned int getGuessHit(const QString& username) const;
00062
00063     float getHitRate(const QString& username);
00064
00065     void renameUser(const QString& oldUsername, const QString& newUsername);
00066
00067     void won(const QString& username);
00068
00069     void lost(const QString& username);
00070
00071     void hit(const QString& username);
00072
00073     void miss(const QString& username);
00074
00075     QJsonObject loadJsonFile(); // Function to load JSON data
00076
00077     public slots:
00078         void show();
00079
00080     signals:
00081         void backToMainMenu();
00082
00083     private slots:
00084         void handleLogin();
00085
00086         void refreshUserDropdown();
00087
00088         void handleCreateAccount();
00089
00090         void showMainMenu();
00091
00092

```

```

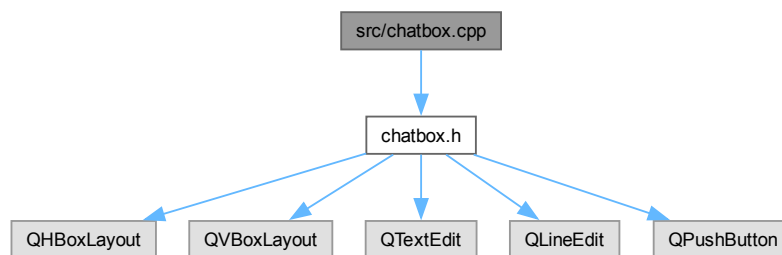
00242 private:
00249     explicit User(QWidget* parent = nullptr);
00250
00255     CreateAccountWindow* createAccountWindow;
00256
00261     QString jsonFilePath = "resources/profile.json";
00262
00267     QPushButton* backButton;
00268
00273     QPushButton* createAccountButton;
00274
00279     QComboBox* usernameComboBox;
00280
00285     QLabel* jsonContentLabel;
00286
00291     QPushButton* loginButton;
00292
00299     void populateUsernameComboBox(const QJsonObject& jsonObject);
00300 };
00301
00302 #endif // USER_H

```

## 5.23 src/chatbox.cpp File Reference

```
#include "chatbox.h"
```

Include dependency graph for chatbox.cpp:

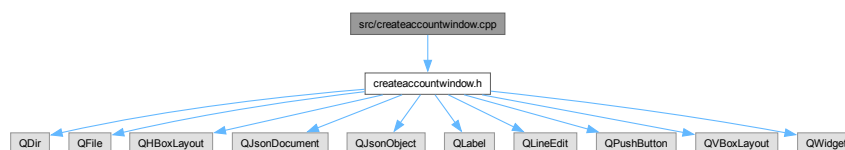


## 5.24 src/createaccountwindow.cpp File Reference

CPP file for the [CreateAccountWindow](#) class which handles user account creation.

```
#include "createaccountwindow.h"
```

Include dependency graph for createaccountwindow.cpp:









### 5.29.1 Detailed Description

CPP file for the [PreGame](#) class which handles the game setup screen.

#### Author

Team 9 - UWO CS 3307

#### Version

0.1

#### Date

2025-03-30

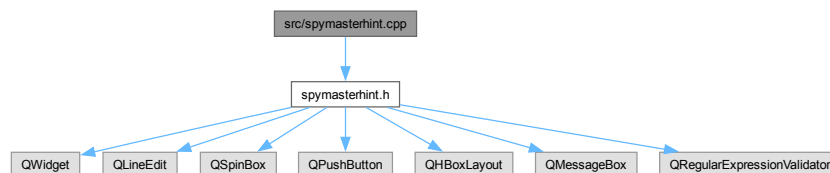
#### Copyright

Copyright (c) 2025

## 5.30 src/spymasterhint.cpp File Reference

```
#include "spymasterhint.h"
```

Include dependency graph for spymasterhint.cpp:

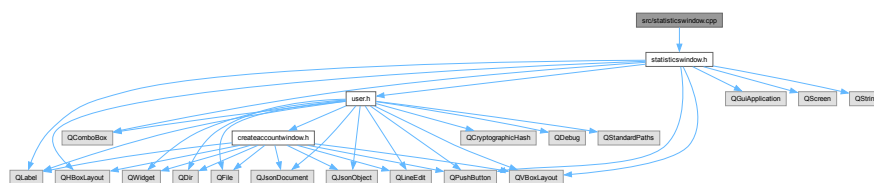


## 5.31 src/statisticswindow.cpp File Reference

The screen to show the user's statistics.

```
#include "statisticswindow.h"
```

Include dependency graph for statisticswindow.cpp:



### 5.31.1 Detailed Description

The screen to show the user's statistics.

#### Author

Team 9 - UWO CS 3307

#### Version

0.1

#### Date

2025-03-30

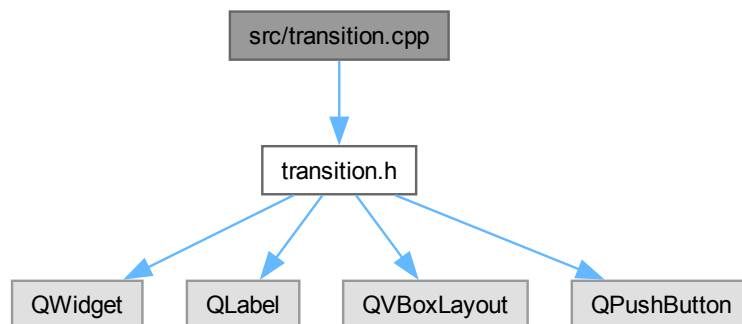
#### Copyright

Copyright (c) 2025

## 5.32 src/transition.cpp File Reference

```
#include "transition.h"
```

Include dependency graph for transition.cpp:



## 5.33 src/tutorial.cpp File Reference

```
#include "tutorial.h"
```

Include dependency graph for tutorial.cpp:

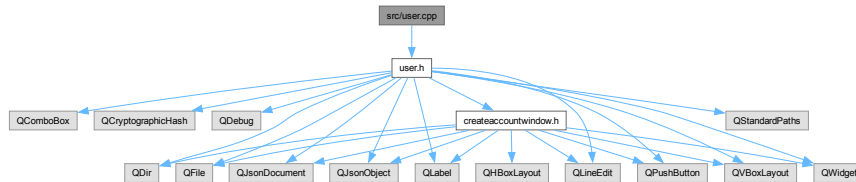


## 5.34 src/user.cpp File Reference

[User](#) class to handle local log in and loading/storing json files.

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

Include dependency graph for user.cpp:



### 5.34.1 Detailed Description

[User](#) class to handle local log in and loading/storing json files.

#### Author

Team 9 - UWO CS 3307

#### Version

0.1

#### Date

2025-03-30

#### Copyright

Copyright (c) 2025



# Index

- ~ChatBox
  - ChatBox, [10](#)
- ~GameBoard
  - GameBoard, [22](#)
- ~MainWindow
  - MainWindow, [34](#)
- ~OperatorGuess
  - OperatorGuess, [39](#)
- ~PreGame
  - PreGame, [43](#)
- ~SpymasterHint
  - SpymasterHint, [49](#)
- ~StatisticsWindow
  - StatisticsWindow, [54](#)
- ~Transition
  - Transition, [58](#)
- ~Tutorial
  - Tutorial, [61](#)
- ~User
  - User, [66](#)
- accountCreated
  - CreateAccountWindow, [15](#)
- addPlayerMessage
  - ChatBox, [10](#)
- addSystemMessage
  - ChatBox, [11](#)
- ASSASSIN
  - GameBoard, [21](#)
- back
  - CreateAccountWindow, [15](#)
- backButton
  - PreGame, [45](#)
  - User, [72](#)
- backToMainButton
  - StatisticsWindow, [55](#)
- backToMainMenu
  - User, [66](#)
- backToMainWindow
  - PreGame, [43](#)
  - StatisticsWindow, [54](#)
- BLUE\_OP
  - GameBoard, [22](#)
- BLUE\_SPY
  - GameBoard, [22](#)
- BLUE\_TEAM
  - ChatBox, [10](#)
  - GameBoard, [21](#)
- blueCardsRemaining
  - GameBoard, [28](#)
- blueOperativeName
  - GameBoard, [28](#)
- blueScoreLabel
  - GameBoard, [28](#)
- blueSpyMasterName
  - GameBoard, [28](#)
- blueTeamLabel
  - GameBoard, [28](#)
- blueTeamLayout
  - PreGame, [45](#)
- blueTeamOperativeComboBox
  - PreGame, [45](#)
- blueTeamSpyMasterComboBox
  - PreGame, [46](#)
- buttonsLayout
  - PreGame, [46](#)
- cards
  - GameBoard, [29](#)
- CardType
  - GameBoard, [21](#)
- centralWidget
  - MainWindow, [35](#)
  - Tutorial, [62](#)
- ChatBox, [8](#)
  - ~ChatBox, [10](#)
  - addPlayerMessage, [10](#)
  - addSystemMessage, [11](#)
  - BLUE\_TEAM, [10](#)
  - ChatBox, [10](#)
  - chatDisplay, [12](#)
  - chatInput, [12](#)
  - clearChat, [11](#)
  - limitReachedMessage, [11](#)
  - massSend, [11](#)
  - playerName, [12](#)
  - RED\_TEAM, [10](#)
  - sendButton, [12](#)
  - sendMessage, [12](#)
  - setPlayerName, [12](#)
  - Team, [9](#)
  - team, [13](#)
- chatBox
  - GameBoard, [29](#)
- chatDisplay
  - ChatBox, [12](#)
- chatInput
  - ChatBox, [12](#)
- checkGameEnd

- GameBoard, 23
- clearChat
  - ChatBox, 11
- clickCount
  - Tutorial, 62
- closeEvent
  - Tutorial, 61
- continueButton
  - Transition, 59
  - Tutorial, 62
- continueClicked
  - Transition, 58
- correspondingNumber
  - GameBoard, 29
- createAccountButton
  - CreateAccountWindow, 17
  - MainWindow, 35
  - PreGame, 46
  - User, 72
- CreateAccountWindow, 13
  - accountCreated, 15
  - back, 15
  - createAccountButton, 17
  - CreateAccountWindow, 15
  - getInstance, 15
  - goBack, 16
  - instance, 17
  - jsonFilePath, 17
  - onCreateAccountClicked, 16
  - previousScreen, 17
  - saveJsonFile, 16
  - setPreviousScreen, 16
  - show, 16
  - statusLabel, 17
  - usernameEdit, 17
- createAccountWindow
  - MainWindow, 35
  - PreGame, 46
  - User, 72
- currentGuesses
  - GameBoard, 29
- currentHint
  - GameBoard, 29
- currentPlayerName
  - GameBoard, 29
- currentPlayerTeam
  - GameBoard, 29
- currentTurn
  - GameBoard, 29
- currentTurnLabel
  - GameBoard, 30
- displayGuess
  - GameBoard, 23
- displayHint
  - GameBoard, 23
- endGame
  - GameBoard, 23
- GameBoard, 18
  - ~GameBoard, 22
  - ASSASSIN, 21
  - BLUE\_OP, 22
  - BLUE\_SPY, 22
  - BLUE\_TEAM, 21
  - blueCardsRemaining, 28
  - blueOperativeName, 28
  - blueScoreLabel, 28
  - blueSpyMasterName, 28
  - blueTeamLabel, 28
  - cards, 29
  - CardType, 21
  - chatBox, 29
  - checkGameEnd, 23
  - correspondingNumber, 29
  - currentGuesses, 29
  - currentHint, 29
  - currentPlayerName, 29
  - currentPlayerTeam, 29
  - currentTurn, 29
  - currentTurnLabel, 30
  - displayGuess, 23
  - displayHint, 23
  - endGame, 23
  - GameBoard, 22
  - gameEnded, 24
  - gameGrid, 30
  - generateGameGrid, 24
  - GRID\_SIZE, 30
  - gridLayout, 30
  - loadWordsFromFile, 24
  - maxGuesses, 30
  - NEUTRAL, 21
  - nextTurn, 24
  - onCardClicked, 25
  - onContinueClicked, 25
  - operatorGuess, 30
  - RED\_OP, 22
  - RED\_SPY, 22
  - RED\_TEAM, 21
  - redCardsRemaining, 30
  - redOperativeName, 30
  - redScoreLabel, 31
  - redSpyMasterName, 31
  - redTeamLabel, 31
  - resetGame, 25
  - setBlueOperativeName, 25
  - setBlueSpyMasterName, 26
  - setRedOperativeName, 26
  - setRedSpyMasterName, 26
  - setupUI, 27
  - show, 27
  - showTransition, 27
  - spymasterHint, 31
  - transition, 31
  - Turn, 21
  - updateScores, 27

- updateTeamLabels, 28
- users, 31
- wordList, 31
- gameBoard
  - PreGame, 46
- GameBoard::Card, 7
  - revealed, 7
  - type, 7
  - word, 7
- gameEnded
  - GameBoard, 24
- gameGrid
  - GameBoard, 30
- gamesPlayedStats
  - StatisticsWindow, 55
- gamesWinRateStats
  - StatisticsWindow, 55
- gamesWinStats
  - StatisticsWindow, 55
- generateGameGrid
  - GameBoard, 24
- getBlueTeamOperativeNickname
  - PreGame, 43
- getBlueTeamSpyMasterNickname
  - PreGame, 43
- getGamesPlayed
  - User, 66
- getGuessHit
  - User, 67
- getGuessTotal
  - User, 67
- getHitRate
  - User, 67
- getInstance
  - CreateAccountWindow, 15
- getRedTeamOperativeNickname
  - PreGame, 44
- getRedTeamSpyMasterNickname
  - PreGame, 44
- getWinRate
  - User, 68
- getWins
  - User, 68
- giveClueButton
  - SpymasterHint, 51
- goBack
  - CreateAccountWindow, 16
- goBackToMain
  - PreGame, 44
  - StatisticsWindow, 54
- GRID\_SIZE
  - GameBoard, 30
- gridLayout
  - GameBoard, 30
- guessHitRateStats
  - StatisticsWindow, 55
- guessHitStats
  - StatisticsWindow, 55
- guessSubmitted
  - OperatorGuess, 39
- guessTotalStats
  - StatisticsWindow, 56
- handleCreateAccount
  - User, 68
- handleGameEnd
  - PreGame, 44
- handleLogin
  - User, 68
- hintLineEdit
  - SpymasterHint, 51
- hintSubmitted
  - SpymasterHint, 50
- hit
  - User, 68
- include/chatbox.h, 73, 75
- include/createaccountwindow.h, 75, 77
- include/gameboard.h, 77, 79
- include/mainwindow.h, 80, 81
- include/operatorguess.h, 82, 83
- include/pregame.h, 84, 86
- include/spymasterhint.h, 87, 88
- include/statisticswindow.h, 89, 90
- include/transition.h, 91, 93
- include/tutorial.h, 93, 94
- include/user.h, 95, 97
- instance
  - CreateAccountWindow, 17
  - User, 69
- jsonContentLabel
  - User, 72
- jsonFilePath
  - CreateAccountWindow, 17
  - User, 72
- label
  - PreGame, 46
- layout
  - MainWindow, 35
  - PreGame, 46
- limitReachedMessage
  - ChatBox, 11
- loadJsonFile
  - User, 69
- loadWordsFromFile
  - GameBoard, 24
- localPlayButton
  - MainWindow, 35
- loginButton
  - User, 72
- lost
  - User, 69
- main
  - main.cpp, 100

- main.cpp
  - main, 100
- MainWindow, 32
  - ~MainWindow, 34
  - centralWidget, 35
  - createAccountButton, 35
  - createAccountWindow, 35
  - layout, 35
  - localPlayButton, 35
  - MainWindow, 34
  - onlinePlayButton, 36
  - openCreateAccount, 34
  - openMultiMain, 34
  - openPreGame, 34
  - openStatsWindow, 34
  - openTutorial, 35
  - preGameWindow, 36
  - showMainWindow, 35
  - statsButton, 36
  - statsWindow, 36
  - titleLabel, 36
  - tutorialButton, 36
  - tutorialWindow, 36
- massSend
  - ChatBox, 11
- maxGuesses
  - GameBoard, 30
- messageLabel
  - Transition, 59
- miss
  - User, 69
- NEUTRAL
  - GameBoard, 21
- nextTurn
  - GameBoard, 24
- numberSpinBox
  - SpymasterHint, 51
- onCardClicked
  - GameBoard, 25
- onContinueClicked
  - GameBoard, 25
  - Tutorial, 62
- onCreateAccountClicked
  - CreateAccountWindow, 16
- onlinePlayButton
  - MainWindow, 36
- openCreateAccount
  - MainWindow, 34
  - PreGame, 44
- openMultiMain
  - MainWindow, 34
- openPreGame
  - MainWindow, 34
- openStatsWindow
  - MainWindow, 34
- openTutorial
  - MainWindow, 35
- OperatorGuess, 37
  - ~OperatorGuess, 39
  - guessSubmitted, 39
  - OperatorGuess, 38
  - reset, 39
  - submitGuess, 39
  - submitGuessButton, 40
- operatorGuess
  - GameBoard, 30
- playerName
  - ChatBox, 12
- populateDropDown
  - StatisticsWindow, 54
- populateUserDropdowns
  - PreGame, 44
- populateUsernameComboBox
  - User, 70
- PreGame, 40
  - ~PreGame, 43
  - backButton, 45
  - backToMainWindow, 43
  - blueTeamLayout, 45
  - blueTeamOperativeComboBox, 45
  - blueTeamSpyMasterComboBox, 46
  - buttonsLayout, 46
  - createAccountButton, 46
  - createAccountWindow, 46
  - gameBoard, 46
  - getBlueTeamOperativeNickname, 43
  - getBlueTeamSpyMasterNickname, 43
  - getRedTeamOperativeNickname, 44
  - getRedTeamSpyMasterNickname, 44
  - goBackToMain, 44
  - handleGameEnd, 44
  - label, 46
  - layout, 46
  - openCreateAccount, 44
  - populateUserDropdowns, 44
  - PreGame, 43
  - redTeamLayout, 46
  - redTeamOperativeComboBox, 47
  - redTeamSpyMasterComboBox, 47
  - show, 45
  - start, 45
  - startButton, 47
  - startGame, 45
  - teamsLayout, 47
  - update, 45
  - usernames, 47
  - users, 47
- preGameWindow
  - MainWindow, 36
- previousScreen
  - CreateAccountWindow, 17
- RED\_OP
  - GameBoard, 22
- RED\_SPY



- GameBoard, 22
- RED\_TEAM
  - ChatBox, 10
  - GameBoard, 21
- redCardsRemaining
  - GameBoard, 30
- redOperativeName
  - GameBoard, 30
- redScoreLabel
  - GameBoard, 31
- redSpyMasterName
  - GameBoard, 31
- redTeamLabel
  - GameBoard, 31
- redTeamLayout
  - PreGame, 46
- redTeamOperativeComboBox
  - PreGame, 47
- redTeamSpyMasterComboBox
  - PreGame, 47
- refreshUserDropdown
  - User, 70
- renameUser
  - User, 70
- reset
  - OperatorGuess, 39
  - SpymasterHint, 50
- resetGame
  - GameBoard, 25
- resetTutorial
  - Tutorial, 62
- revealed
  - GameBoard::Card, 7
- saveJsonFile
  - CreateAccountWindow, 16
- sendButton
  - ChatBox, 12
- sendMessage
  - ChatBox, 12
- setBlueOperativeName
  - GameBoard, 25
- setBlueSpyMasterName
  - GameBoard, 26
- setMessage
  - Transition, 58
- setPlayerName
  - ChatBox, 12
- setPreviousScreen
  - CreateAccountWindow, 16
- setRedOperativeName
  - GameBoard, 26
- setRedSpyMasterName
  - GameBoard, 26
- setUpUI
  - GameBoard, 27
- show
  - CreateAccountWindow, 16
  - GameBoard, 27
  - PreGame, 45
  - StatisticsWindow, 54
  - User, 70
- showMainMenu
  - User, 70
- showMainWindow
  - MainWindow, 35
- showTransition
  - GameBoard, 27
- showUserStats
  - StatisticsWindow, 55
- showUserStatsButton
  - StatisticsWindow, 56
- SpymasterHint, 48
  - ~SpymasterHint, 49
  - giveClueButton, 51
  - hintLineEdit, 51
  - hintSubmitted, 50
  - numberSpinBox, 51
  - reset, 50
  - SpymasterHint, 49
  - submitHint, 50
  - textToUppercase, 50
  - textValidator, 51
  - updateButtonClickable, 51
- spygameHint
  - GameBoard, 31
- src/chatbox.cpp, 98
- src/createaccountwindow.cpp, 98
- src/gameboard.cpp, 99
- src/main.cpp, 99
- src/mainwindow.cpp, 100
- src/operatorguess.cpp, 100
- src/pregame.cpp, 100
- src/spymasterhint.cpp, 101
- src/statisticswindow.cpp, 101
- src/transition.cpp, 102
- src/tutorial.cpp, 102
- src/user.cpp, 103
- start
  - PreGame, 45
- startButton
  - PreGame, 47
- startGame
  - PreGame, 45
- StatisticsWindow, 52
  - ~StatisticsWindow, 54
  - backToMainButton, 55
  - backToMainWindow, 54
  - gamesPlayedStats, 55
  - gamesWinRateStats, 55
  - gamesWinStats, 55
  - goBackToMain, 54
  - guessHitRateStats, 55
  - guessHitStats, 55
  - guessTotalStats, 56
  - populateDropDown, 54
  - show, 54

- showUserStats, 55
  - showUserStatsButton, 56
  - StatisticsWindow, 54
  - username, 56
  - usernameComboBox, 56
  - usernameTitle, 56
  - users, 56
- statsButton
  - MainWindow, 36
- statsWindow
  - MainWindow, 36
- statusLabel
  - CreateAccountWindow, 17
- submitGuess
  - OperatorGuess, 39
- submitGuessButton
  - OperatorGuess, 40
- submitHint
  - SpymasterHint, 50
- Team
  - ChatBox, 9
- team
  - ChatBox, 13
- teamsLayout
  - PreGame, 47
- textBox
  - Tutorial, 63
- textToUppercase
  - SpymasterHint, 50
- textValidator
  - SpymasterHint, 51
- titleLabel
  - MainWindow, 36
  - Tutorial, 63
- Transition, 57
  - ~Transition, 58
  - continueButton, 59
  - continueClicked, 58
  - messageLabel, 59
  - setMessage, 58
  - Transition, 58
- transition
  - GameBoard, 31
- Turn
  - GameBoard, 21
- Tutorial, 59
  - ~Tutorial, 61
  - centralWidget, 62
  - clickCount, 62
  - closeEvent, 61
  - continueButton, 62
  - onContinueClicked, 62
  - resetTutorial, 62
  - textBox, 63
  - titleLabel, 63
  - Tutorial, 61
  - tutorialClosed, 62
  - updateContinueButtonPosition, 62
- tutorialButton
  - MainWindow, 36
- tutorialClosed
  - Tutorial, 62
- tutorialWindow
  - MainWindow, 36
- type
  - GameBoard::Card, 7
- update
  - PreGame, 45
- updateButtonClickable
  - SpymasterHint, 51
- updateContinueButtonPosition
  - Tutorial, 62
- updateGamesPlayed
  - User, 70
- updateGuessHit
  - User, 71
- updateGuessTotal
  - User, 71
- updateScores
  - GameBoard, 27
- updateTeamLabels
  - GameBoard, 28
- updateWins
  - User, 71
- User, 63
  - ~User, 66
  - backButton, 72
  - backToMainMenu, 66
  - createAccountButton, 72
  - createAccountWindow, 72
  - getGamesPlayed, 66
  - getGuessHit, 67
  - getGuessTotal, 67
  - getHitRate, 67
  - getWinRate, 68
  - getWins, 68
  - handleCreateAccount, 68
  - handleLogin, 68
  - hit, 68
  - instance, 69
  - jsonContentLabel, 72
  - jsonFilePath, 72
  - loadJsonFile, 69
  - loginButton, 72
  - lost, 69
  - miss, 69
  - populateUsernameComboBox, 70
  - refreshUserDropdown, 70
  - renameUser, 70
  - show, 70
  - showMainMenu, 70
  - updateGamesPlayed, 70
  - updateGuessHit, 71
  - updateGuessTotal, 71
  - updateWins, 71
  - User, 66

- usernameComboBox, [72](#)
  - won, [71](#)
- username
  - StatisticsWindow, [56](#)
- usernameComboBox
  - StatisticsWindow, [56](#)
  - User, [72](#)
- usernameEdit
  - CreateAccountWindow, [17](#)
- usernames
  - PreGame, [47](#)
- usernameTitle
  - StatisticsWindow, [56](#)
- users
  - GameBoard, [31](#)
  - PreGame, [47](#)
  - StatisticsWindow, [56](#)
- won
  - User, [71](#)
- word
  - GameBoard::Card, [7](#)
- wordList
  - GameBoard, [31](#)