

## 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 MultiBoard::Card Struct Reference	8
4.2.1 Detailed Description	8
4.2.2 Member Data Documentation	8
4.2.2.1 revealed	8
4.2.2.2 type	8
4.2.2.3 word	8
4.3 ChatBox Class Reference	9
4.3.1 Detailed Description	10
4.3.2 Member Enumeration Documentation	10
4.3.2.1 Team	10
4.3.3 Constructor & Destructor Documentation	11
4.3.3.1 ChatBox()	11
4.3.3.2 ~ChatBox()	11
4.3.4 Member Function Documentation	11
4.3.4.1 addPlayerMessage()	11
4.3.4.2 addSystemMessage()	12
4.3.4.3 clearChat()	12
4.3.4.4 limitReachedMessage()	12
4.3.4.5 massSend	12
4.3.4.6 sendMessage	13
4.3.4.7 setPlayerName()	13
4.3.5 Member Data Documentation	13
4.3.5.1 chatDisplay	13
4.3.5.2 chatInput	13
4.3.5.3 playerName	13
4.3.5.4 sendButton	14
4.3.5.5 team	14

4.4 CreateAccountWindow Class Reference	14
4.4.1 Detailed Description	16
4.4.2 Constructor & Destructor Documentation	16
4.4.2.1 CreateAccountWindow()	16
4.4.3 Member Function Documentation	16
4.4.3.1 accountCreated	16
4.4.3.2 back	16
4.4.3.3 getInstance()	16
4.4.3.4 goBack	17
4.4.3.5 onCreateAccountClicked	17
4.4.3.6 saveJsonFile()	17
4.4.3.7 setPreviousScreen()	17
4.4.3.8 show	17
4.4.4 Member Data Documentation	18
4.4.4.1 createAccountButton	18
4.4.4.2 instance	18
4.4.4.3 jsonFilePath	18
4.4.4.4 previousScreen	18
4.4.4.5 statusLabel	18
4.4.4.6 usernameEdit	18
4.5 GameBoard Class Reference	19
4.5.1 Detailed Description	22
4.5.2 Member Enumeration Documentation	22
4.5.2.1 CardType	22
4.5.2.2 Turn	22
4.5.3 Constructor & Destructor Documentation	23
4.5.3.1 GameBoard()	23
4.5.3.2 ~GameBoard()	23
4.5.4 Member Function Documentation	24
4.5.4.1 checkGameEnd()	24
4.5.4.2 displayGuess	24
4.5.4.3 displayHint	24
4.5.4.4 endGame()	24
4.5.4.5 gameEnded	25
4.5.4.6 generateGameGrid()	25
4.5.4.7 loadWordsFromFile()	25
4.5.4.8 nextTurn()	26
4.5.4.9 onCardClicked()	26
4.5.4.10 onContinueClicked()	26
4.5.4.11 resetGame()	26
4.5.4.12 setBlueOperativeName()	26
4.5.4.13 setBlueSpyMasterName()	27

4.5.4.14 setRedOperativeName()	27
4.5.4.15 setRedSpyMasterName()	27
4.5.4.16 setupUI()	28
4.5.4.17 show	28
4.5.4.18 showTransition()	28
4.5.4.19 updateScores()	29
4.5.4.20 updateTeamLabels()	29
4.5.5 Member Data Documentation	29
4.5.5.1 blueCardsRemaining	29
4.5.5.2 blueOperativeName	29
4.5.5.3 blueScoreLabel	29
4.5.5.4 blueSpyMasterName	29
4.5.5.5 blueTeamLabel	30
4.5.5.6 cards	30
4.5.5.7 chatBox	30
4.5.5.8 correspondingNumber	30
4.5.5.9 currentGuesses	30
4.5.5.10 currentHint	30
4.5.5.11 currentPlayerName	30
4.5.5.12 currentPlayerTeam	30
4.5.5.13 currentTurn	31
4.5.5.14 currentTurnLabel	31
4.5.5.15 gameGrid	31
4.5.5.16 GRID_SIZE	31
4.5.5.17 gridLayout	31
4.5.5.18 maxGuesses	31
4.5.5.19 operatorGuess	31
4.5.5.20 redCardsRemaining	31
4.5.5.21 redOperativeName	32
4.5.5.22 redScoreLabel	32
4.5.5.23 redSpyMasterName	32
4.5.5.24 redTeamLabel	32
4.5.5.25 spymasterHint	32
4.5.5.26 transition	32
4.5.5.27 users	32
4.5.5.28 wordList	33
4.6 MainWindow Class Reference	33
4.6.1 Detailed Description	35
4.6.2 Constructor & Destructor Documentation	35
4.6.2.1 MainWindow()	35
4.6.2.2 ~MainWindow()	35
4.6.3 Member Function Documentation	35

---

4.6.3.1 openCreateAccount . . . . .	35
4.6.3.2 openMultiMain . . . . .	35
4.6.3.3 openOnlineGame . . . . .	35
4.6.3.4 openPreGame . . . . .	36
4.6.3.5 openStatsWindow . . . . .	36
4.6.3.6 openTutorial . . . . .	36
4.6.3.7 showMainWindow . . . . .	36
4.6.4 Member Data Documentation . . . . .	36
4.6.4.1 centralWidget . . . . .	36
4.6.4.2 createAccountButton . . . . .	36
4.6.4.3 createAccountWindow . . . . .	36
4.6.4.4 layout . . . . .	37
4.6.4.5 localPlayButton . . . . .	37
4.6.4.6 multiMain . . . . .	37
4.6.4.7 onlineGameWindow . . . . .	37
4.6.4.8 onlinePlayButton . . . . .	37
4.6.4.9 preGameWindow . . . . .	37
4.6.4.10 statsButton . . . . .	37
4.6.4.11 statsWindow . . . . .	37
4.6.4.12 titleLabel . . . . .	38
4.6.4.13 tutorialButton . . . . .	38
4.6.4.14 tutorialWindow . . . . .	38
4.7 MultiBoard Class Reference . . . . .	38
4.7.1 Detailed Description . . . . .	42
4.7.2 Member Enumeration Documentation . . . . .	42
4.7.2.1 CardType . . . . .	42
4.7.2.2 Turn . . . . .	42
4.7.3 Constructor & Destructor Documentation . . . . .	43
4.7.3.1 MultiBoard() . . . . .	43
4.7.4 Member Function Documentation . . . . .	43
4.7.4.1 advanceTurn() . . . . .	43
4.7.4.2 advanceTurnSpymaster() . . . . .	44
4.7.4.3 checkGameEnd() . . . . .	44
4.7.4.4 displayHint() . . . . .	44
4.7.4.5 endGame() . . . . .	44
4.7.4.6 generateGameGrid() . . . . .	45
4.7.4.7 getColorStyle() . . . . .	45
4.7.4.8 getMyTeam() . . . . .	46
4.7.4.9 goBack . . . . .	46
4.7.4.10 handleNewConnection . . . . .	46
4.7.4.11 handleTileClick . . . . .	46
4.7.4.12 initializeBoardColors() . . . . .	47

4.7.4.13 initializeWords()	47
4.7.4.14 isMyTurn()	47
4.7.4.15 loadWordsFromFile()	47
4.7.4.16 processChatMessage()	47
4.7.4.17 processMessage	48
4.7.4.18 revealTile()	48
4.7.4.19 sendInitialGameState()	49
4.7.4.20 sendToAll()	49
4.7.4.21 setupBoard()	49
4.7.4.22 setupUI()	49
4.7.4.23 socketDisconnected	50
4.7.4.24 updateTurnDisplay()	50
4.7.5 Member Data Documentation	50
4.7.5.1 blueCardsRemaining	50
4.7.5.2 blueCardText	50
4.7.5.3 cards	50
4.7.5.4 chatBox	50
4.7.5.5 correspondingNumber	51
4.7.5.6 currentHint	51
4.7.5.7 gameGrid	51
4.7.5.8 gameVerticalLayout	51
4.7.5.9 GRID_SIZE	51
4.7.5.10 guess	51
4.7.5.11 hint	51
4.7.5.12 m_clients	51
4.7.5.13 m_clientSocket	52
4.7.5.14 m_currentRole	52
4.7.5.15 m_currentTurnIndex	52
4.7.5.16 m_currentUsername	52
4.7.5.17 m_grid	52
4.7.5.18 m_isHost	52
4.7.5.19 m_playerInfoLabel	52
4.7.5.20 m_playerRoles	52
4.7.5.21 m_pregame	53
4.7.5.22 m_server	53
4.7.5.23 m_tileColors	53
4.7.5.24 m_tiles	53
4.7.5.25 m_turnLabel	53
4.7.5.26 m_turnOrder	53
4.7.5.27 m_words	53
4.7.5.28 main	53
4.7.5.29 mainLayout	54

4.7.5.30 redCardsRemaining . . . . .	54
4.7.5.31 redCardText . . . . .	54
4.7.5.32 users . . . . .	54
4.7.5.33 wordList . . . . .	54
4.8 MultiMain Class Reference . . . . .	55
4.8.1 Detailed Description . . . . .	57
4.8.2 Constructor & Destructor Documentation . . . . .	57
4.8.2.1 MultiMain() . . . . .	57
4.8.2.2 ~MultiMain() . . . . .	57
4.8.3 Member Function Documentation . . . . .	57
4.8.3.1 backToMainWindow . . . . .	57
4.8.3.2 enterPregameAsClient . . . . .	57
4.8.3.3 enterPregameAsHost . . . . .	58
4.8.3.4 onConnected . . . . .	58
4.8.3.5 onCreateRoomClicked . . . . .	58
4.8.3.6 onDisconnected . . . . .	59
4.8.3.7 onJoinRoomClicked . . . . .	59
4.8.3.8 onNewConnection . . . . .	59
4.8.3.9 openMainWindow . . . . .	59
4.8.3.10 processTextMessage . . . . .	59
4.8.3.11 sendLobbyListToAll() . . . . .	60
4.8.3.12 showMainWindow() . . . . .	60
4.8.3.13 socketDisconnected . . . . .	60
4.8.3.14 updateLobbyList() . . . . .	61
4.8.4 Member Data Documentation . . . . .	61
4.8.4.1 backButton . . . . .	61
4.8.4.2 createRoomButton . . . . .	61
4.8.4.3 joinRoomButton . . . . .	61
4.8.4.4 m_clients . . . . .	61
4.8.4.5 m_clientSocket . . . . .	61
4.8.4.6 m_server . . . . .	62
4.8.4.7 m_username . . . . .	62
4.8.4.8 m_usernames . . . . .	62
4.8.4.9 titleLabel . . . . .	62
4.9 MultiPregame Class Reference . . . . .	62
4.9.1 Detailed Description . . . . .	64
4.9.2 Constructor & Destructor Documentation . . . . .	65
4.9.2.1 MultiPregame() [1/2] . . . . .	65
4.9.2.2 MultiPregame() [2/2] . . . . .	65
4.9.2.3 ~MultiPregame() . . . . .	66
4.9.3 Member Function Documentation . . . . .	66
4.9.3.1 backToMultiMain . . . . .	66



4.9.3.2 clearUI()	66
4.9.3.3 enterPregameAsClient	66
4.9.3.4 enterPregameAsHost	66
4.9.3.5 gameStarted()	67
4.9.3.6 handleRoleSelection()	67
4.9.3.7 onNewConnection	68
4.9.3.8 processMessage	68
4.9.3.9 resetUIState()	68
4.9.3.10 sendLobbyUpdate()	68
4.9.3.11 setupUI()	69
4.9.3.12 showPregame()	69
4.9.3.13 socketDisconnected	69
4.9.3.14 startGame	69
4.9.4 Member Data Documentation	70
4.9.4.1 m_checked	70
4.9.4.2 m_clients	70
4.9.4.3 m_clientSocket	70
4.9.4.4 m_isHost	70
4.9.4.5 m_roles	70
4.9.4.6 m_server	70
4.9.4.7 m_username	70
4.9.4.8 m_usernames	71
4.9.4.9 playerList	71
4.10 OperatorGuess Class Reference	71
4.10.1 Detailed Description	72
4.10.2 Constructor & Destructor Documentation	72
4.10.2.1 OperatorGuess()	72
4.10.2.2 ~OperatorGuess()	73
4.10.3 Member Function Documentation	73
4.10.3.1 guessSubmitted	73
4.10.3.2 reset()	73
4.10.3.3 submitGuess	74
4.10.4 Member Data Documentation	74
4.10.4.1 submitGuessButton	74
4.11 PreGame Class Reference	74
4.11.1 Detailed Description	77
4.11.2 Constructor & Destructor Documentation	77
4.11.2.1 PreGame()	77
4.11.2.2 ~PreGame()	77
4.11.3 Member Function Documentation	77
4.11.3.1 backToMainWindow	77
4.11.3.2 getBlueTeamOperativeNickname()	77

4.11.3.3 getBlueTeamSpyMasterNickname()	78
4.11.3.4 getRedTeamOperativeNickname()	78
4.11.3.5 getRedTeamSpyMasterNickname()	78
4.11.3.6 goBackToMain	78
4.11.3.7 handleGameEnd	78
4.11.3.8 openCreateAccount	78
4.11.3.9 populateUserDropdowns()	79
4.11.3.10 show	79
4.11.3.11 start	79
4.11.3.12 startGame	79
4.11.3.13 update	79
4.11.4 Member Data Documentation	79
4.11.4.1 backButton	79
4.11.4.2 blueTeamLayout	79
4.11.4.3 blueTeamOperativeComboBox	80
4.11.4.4 blueTeamSpyMasterComboBox	80
4.11.4.5 buttonsLayout	80
4.11.4.6 createAccountButton	80
4.11.4.7 createAccountWindow	80
4.11.4.8 gameBoard	80
4.11.4.9 label	80
4.11.4.10 layout	80
4.11.4.11 redTeamLayout	81
4.11.4.12 redTeamOperativeComboBox	81
4.11.4.13 redTeamSpyMasterComboBox	81
4.11.4.14 startButton	81
4.11.4.15 teamsLayout	81
4.11.4.16 usernames	81
4.11.4.17 users	81
4.12 SpymasterHint Class Reference	82
4.12.1 Detailed Description	83
4.12.2 Constructor & Destructor Documentation	83
4.12.2.1 SpymasterHint()	83
4.12.2.2 ~SpymasterHint()	84
4.12.3 Member Function Documentation	84
4.12.3.1 hintSubmitted	84
4.12.3.2 reset()	84
4.12.3.3 submitHint	84
4.12.3.4 textToUppercase	84
4.12.3.5 updateButtonClickable	85
4.12.4 Member Data Documentation	85
4.12.4.1 giveClueButton	85

4.12.4.2 hintLineEdit . . . . .	85
4.12.4.3 numberSpinBox . . . . .	85
4.12.4.4 textValidator . . . . .	85
4.13 StatisticsWindow Class Reference . . . . .	86
4.13.1 Detailed Description . . . . .	88
4.13.2 Constructor & Destructor Documentation . . . . .	88
4.13.2.1 StatisticsWindow() . . . . .	88
4.13.2.2 ~StatisticsWindow() . . . . .	88
4.13.3 Member Function Documentation . . . . .	88
4.13.3.1 backToMainWindow . . . . .	88
4.13.3.2 goBackToMain . . . . .	88
4.13.3.3 populateDropDown() . . . . .	88
4.13.3.4 show . . . . .	89
4.13.3.5 showUserStats . . . . .	89
4.13.4 Member Data Documentation . . . . .	89
4.13.4.1 backToMainButton . . . . .	89
4.13.4.2 gamesPlayedStats . . . . .	89
4.13.4.3 gamesWinRateStats . . . . .	89
4.13.4.4 gamesWinStats . . . . .	89
4.13.4.5 guessHitRateStats . . . . .	89
4.13.4.6 guessHitStats . . . . .	90
4.13.4.7 guessTotalStats . . . . .	90
4.13.4.8 showUserStatsButton . . . . .	90
4.13.4.9 username . . . . .	90
4.13.4.10 usernameComboBox . . . . .	90
4.13.4.11 usernameTitle . . . . .	90
4.13.4.12 users . . . . .	90
4.14 Transition Class Reference . . . . .	91
4.14.1 Detailed Description . . . . .	92
4.14.2 Constructor & Destructor Documentation . . . . .	92
4.14.2.1 Transition() . . . . .	92
4.14.2.2 ~Transition() . . . . .	92
4.14.3 Member Function Documentation . . . . .	92
4.14.3.1 continueClicked . . . . .	92
4.14.3.2 setMessage() . . . . .	92
4.14.4 Member Data Documentation . . . . .	93
4.14.4.1 continueButton . . . . .	93
4.14.4.2 messageLabel . . . . .	93
4.15 Tutorial Class Reference . . . . .	93
4.15.1 Detailed Description . . . . .	95
4.15.2 Constructor & Destructor Documentation . . . . .	95
4.15.2.1 Tutorial() . . . . .	95

4.15.2.2 ~Tutorial()	95
4.15.3 Member Function Documentation	95
4.15.3.1 closeEvent()	95
4.15.3.2 onContinueClicked	96
4.15.3.3 resetTutorial()	96
4.15.3.4 tutorialClosed	96
4.15.3.5 updateContinueButtonPosition()	96
4.15.4 Member Data Documentation	96
4.15.4.1 centralWidget	96
4.15.4.2 clickCount	96
4.15.4.3 continueButton	97
4.15.4.4 textBox	97
4.15.4.5 titleLabel	97
4.16 User Class Reference	97
4.16.1 Detailed Description	100
4.16.2 Constructor & Destructor Documentation	100
4.16.2.1 ~User()	100
4.16.2.2 User()	100
4.16.3 Member Function Documentation	100
4.16.3.1 backToMainMenu	100
4.16.3.2 getGamesPlayed()	100
4.16.3.3 getGuessHit()	101
4.16.3.4 getGuessTotal()	101
4.16.3.5 getHitRate()	101
4.16.3.6 getWinRate()	102
4.16.3.7 getWins()	102
4.16.3.8 handleCreateAccount	102
4.16.3.9 handleLogin	102
4.16.3.10 hit()	102
4.16.3.11 instance()	103
4.16.3.12 loadJsonFile()	103
4.16.3.13 lost()	103
4.16.3.14 miss()	103
4.16.3.15 populateUsernameComboBox()	104
4.16.3.16 refreshUserDropdown	104
4.16.3.17 renameUser()	104
4.16.3.18 show	104
4.16.3.19 showMainMenu	104
4.16.3.20 updateGamesPlayed()	104
4.16.3.21 updateGuessHit()	105
4.16.3.22 updateGuessTotal()	105
4.16.3.23 updateWins()	105

4.16.3.24 won()	105
4.16.4 Member Data Documentation	106
4.16.4.1 backButton	106
4.16.4.2 createAccountButton	106
4.16.4.3 createAccountWindow	106
4.16.4.4 jsonContentLabel	106
4.16.4.5 jsonFilePath	106
4.16.4.6 loginButton	106
4.16.4.7 usernameComboBox	106
<b>5 File Documentation</b>	<b>107</b>
5.1 include/chatbox.h File Reference	107
5.1.1 Detailed Description	108
5.2 chatbox.h	109
5.3 include/createaccountwindow.h File Reference	109
5.3.1 Detailed Description	110
5.4 createaccountwindow.h	111
5.5 include/gameboard.h File Reference	111
5.5.1 Detailed Description	112
5.6 gameboard.h	113
5.7 include/mainwindow.h File Reference	114
5.7.1 Detailed Description	115
5.8 mainwindow.h	115
5.9 include/Multiplayer/multiboard.h File Reference	116
5.9.1 Detailed Description	117
5.10 multiboard.h	118
5.11 include/Multiplayer/multimain.h File Reference	120
5.11.1 Detailed Description	121
5.12 multimain.h	121
5.13 include/Multiplayer/multipregame.h File Reference	122
5.13.1 Detailed Description	123
5.14 multipregame.h	123
5.15 include/operatorguess.h File Reference	124
5.15.1 Detailed Description	125
5.16 operatorguess.h	125
5.17 include/pregame.h File Reference	126
5.17.1 Detailed Description	127
5.18 pregame.h	128
5.19 include/spymasterhint.h File Reference	129
5.19.1 Detailed Description	130
5.20 spymasterhint.h	130
5.21 include/statisticswindow.h File Reference	131

5.21.1 Detailed Description . . . . .	131
5.22 statisticswindow.h . . . . .	132
5.23 include/transition.h File Reference . . . . .	133
5.23.1 Detailed Description . . . . .	134
5.24 transition.h . . . . .	135
5.25 include/tutorial.h File Reference . . . . .	135
5.25.1 Detailed Description . . . . .	136
5.26 tutorial.h . . . . .	136
5.27 include/user.h File Reference . . . . .	137
5.27.1 Detailed Description . . . . .	138
5.28 user.h . . . . .	139
5.29 src/chatbox.cpp File Reference . . . . .	140
5.30 src/createaccountwindow.cpp File Reference . . . . .	140
5.30.1 Detailed Description . . . . .	141
5.31 src/gameboard.cpp File Reference . . . . .	141
5.32 src/main.cpp File Reference . . . . .	141
5.32.1 Function Documentation . . . . .	142
5.32.1.1 main() . . . . .	142
5.33 src/mainwindow.cpp File Reference . . . . .	142
5.34 src/Multiplayer/multiboard.cpp File Reference . . . . .	142
5.35 src/Multiplayer/multimain.cpp File Reference . . . . .	142
5.36 src/Multiplayer/multipregame.cpp File Reference . . . . .	142
5.37 src/operatorguess.cpp File Reference . . . . .	143
5.38 src/pregame.cpp File Reference . . . . .	143
5.38.1 Detailed Description . . . . .	143
5.39 src/spymasterhint.cpp File Reference . . . . .	144
5.40 src/statisticswindow.cpp File Reference . . . . .	144
5.40.1 Detailed Description . . . . .	144
5.41 src/transition.cpp File Reference . . . . .	145
5.42 src/tutorial.cpp File Reference . . . . .	145
5.43 src/user.cpp File Reference . . . . .	145
5.43.1 Detailed Description . . . . .	146
<b>Index</b>	<b>147</b>

# Chapter 1

## Hierarchical Index

### 1.1 Class Hierarchy

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

GameBoard::Card . . . . .	7
MultiBoard::Card . . . . .	8
QMainWindow	
MainWindow . . . . .	33
Tutorial . . . . .	93
QWidget	
ChatBox . . . . .	9
CreateAccountWindow . . . . .	14
GameBoard . . . . .	19
MultiBoard . . . . .	38
MultiMain . . . . .	55
MultiPregame . . . . .	62
OperatorGuess . . . . .	71
PreGame . . . . .	74
SpymasterHint . . . . .	82
StatisticsWindow . . . . .	86
Transition . . . . .	91
User . . . . .	97





## 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="#">MultiBoard::Card</a>	Structure representing a card on the game board . . . . .	8
<a href="#">ChatBox</a>	A widget for the chat feature in the game . . . . .	9
<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 . . . . .	14
<a href="#">GameBoard</a>	A class representing the game board for the Spy Master game . . . . .	19
<a href="#">MainWindow</a>	The main application window . . . . .	33
<a href="#">MultiBoard</a>	A widget that implements the multiplayer game board for a team-based word guessing game . . . . .	38
<a href="#">MultiMain</a>	A widget that implements the main multiplayer lobby for creating and joining game rooms . . . . .	55
<a href="#">MultiPregame</a>	A widget that implements the pre-game lobby for multiplayer games . . . . .	62
<a href="#">OperatorGuess</a>	A widget that provides the interface for operators to submit guesses during gameplay . . . . .	71
<a href="#">PreGame</a>	Interface for setting up a new game This includes selecting players for each team and role before starting the game . . . . .	74
<a href="#">SpymasterHint</a>	A widget for the spymaster to input a hint and the number of words associated with it . . . . .	82
<a href="#">StatisticsWindow</a>	The class that shows the Statistics screen Displays game statistics for selected users including win rates and guess accuracy . . . . .	86
<a href="#">Transition</a>	A widget for displaying a transition message and a button to continue . . . . .	91
<a href="#">Tutorial</a>	The tutorial window that guides users through the game mechanics . . . . .	93
<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 . . . . .	97



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

include/chatbox.h	Header file for the <a href="#">ChatBox</a> class, which provides a UI for the chat feature in the game . . . . .	107
include/createaccountwindow.h	Header file for the <a href="#">CreateAccountWindow</a> class which handles user account creation . . . . .	109
include/gameboard.h	Header file for the <a href="#">GameBoard</a> class, which implements a game board for the Spy Master game . . . . .	111
include/mainwindow.h	Declaration of the <a href="#">MainWindow</a> class . . . . .	114
include/operatorguess.h	Header file for the <a href="#">OperatorGuess</a> class, which handles operator guessing interface . . . . .	124
include/pregame.h	Header file for the <a href="#">PreGame</a> class which handles the game setup screen . . . . .	126
include/spymasterhint.h	Header file for the <a href="#">SpymasterHint</a> class, which provides a UI for the spymaster to give hints . . . . .	129
include/statisticswindow.h	The screen to show the user's statistics . . . . .	131
include/transition.h	Header file for the <a href="#">Transition</a> class, which provides a UI for transitions between game states . . . . .	133
include/tutorial.h	Declaration of the <a href="#">Tutorial</a> class . . . . .	135
include/user.h	<a href="#">User</a> class to handle local log in and loading/storing json files . . . . .	137
include/Multiplayer/multiboard.h	Header file for the <a href="#">MultiBoard</a> class, which implements a multiplayer game board . . . . .	116
include/Multiplayer/multimain.h	Header file for the <a href="#">MultiMain</a> class, which implements the main multiplayer game lobby interface . . . . .	120
include/Multiplayer/multipregame.h	Header file for the <a href="#">MultiPregame</a> class, which implements the pre-game lobby for multiplayer games . . . . .	122
src/chatbox.cpp	. . . . .	140
src/createaccountwindow.cpp	CPP file for the <a href="#">CreateAccountWindow</a> class which handles user account creation . . . . .	140
src/gameboard.cpp	. . . . .	141
src/main.cpp	. . . . .	141
src/mainwindow.cpp	. . . . .	142

src/operatorguess.cpp . . . . .	143
src/pregame.cpp	
CPP file for the <a href="#">PreGame</a> class which handles the game setup screen . . . . .	143
src/spymasterhint.cpp . . . . .	144
src/statisticswindow.cpp	
The screen to show the user's statistics . . . . .	144
src/transition.cpp . . . . .	145
src/tutorial.cpp . . . . .	145
src/user.cpp	
<a href="#">User</a> class to handle local log in and loading/storing json files . . . . .	145
src/Multiplayer/multiboard.cpp . . . . .	142
src/Multiplayer/multimain.cpp . . . . .	142
src/Multiplayer/multipregame.cpp . . . . .	142

## 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 MultiBoard::Card Struct Reference

Structure representing a card on the game board.

```
#include <multiboard.h>
```

### Public Attributes

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

### 4.2.1 Detailed Description

Structure representing a card on the game board.

### 4.2.2 Member Data Documentation

#### 4.2.2.1 revealed

```
bool MultiBoard::Card::revealed
```

Whether the card has been revealed

#### 4.2.2.2 type

```
CardType MultiBoard::Card::type
```

The type/team the card belongs to

#### 4.2.2.3 word

```
QString MultiBoard::Card::word
```

The word displayed on the card

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

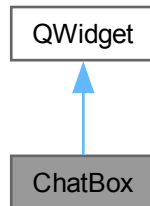
- include/Multiplayer/[multiboard.h](#)

## 4.3 ChatBox Class Reference

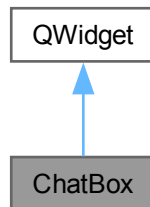
A widget for the chat feature in the game.

```
#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.*

### 4.3.1 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

### 4.3.2 Member Enumeration Documentation

#### 4.3.2.1 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	

### 4.3.3 Constructor & Destructor Documentation

#### 4.3.3.1 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.

#### 4.3.3.2 ~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.

### 4.3.4 Member Function Documentation

#### 4.3.4.1 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.

**4.3.4.2 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).

**4.3.4.3 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.

**4.3.4.4 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.

**4.3.4.5 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.

**4.3.4.6 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.

**4.3.4.7 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.
-------------	-------------------------

**4.3.5 Member Data Documentation****4.3.5.1 chatDisplay**

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

The text edit widget for displaying chat messages.

**4.3.5.2 chatInput**

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

The line edit widget for inputting chat messages.

**4.3.5.3 playerName**

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

The name of the player using this chat box.

#### 4.3.5.4 sendButton

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

The button to send chat messages.

#### 4.3.5.5 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:

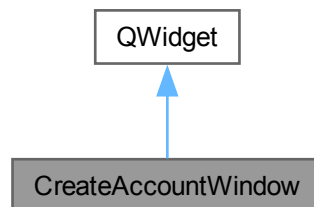
- [include/chatbox.h](#)
- [src/chatbox.cpp](#)

## 4.4 CreateAccountWindow Class Reference

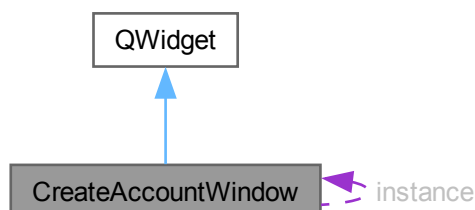
The [CreateAccountWindow](#) class provides a singleton interface for creating new user accounts This window allows users to input a username and creates a profile JSON file for the new account.

```
#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.*

### 4.4.1 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.

### 4.4.2 Constructor & Destructor Documentation

#### 4.4.2.1 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
---------------	--

### 4.4.3 Member Function Documentation

#### 4.4.3.1 accountCreated

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

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

#### 4.4.3.2 back

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

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

#### 4.4.3.3 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

**4.4.3.4 goBack**

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

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

**4.4.3.5 onCreateAccountClicked**

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

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

**4.4.3.6 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
-----------------	----------------------------------

**4.4.3.7 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
-----------------	------------------------------------

**4.4.3.8 show**

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

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

## 4.4.4 Member Data Documentation

### 4.4.4.1 createAccountButton

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

Button to submit account creation request.

### 4.4.4.2 instance

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

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

### 4.4.4.3 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.

### 4.4.4.4 previousScreen

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

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

### 4.4.4.5 statusLabel

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

Label to display status messages and error feedback.

### 4.4.4.6 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](#)

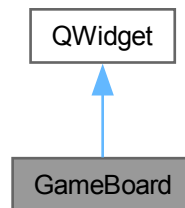


## 4.5 GameBoard Class Reference

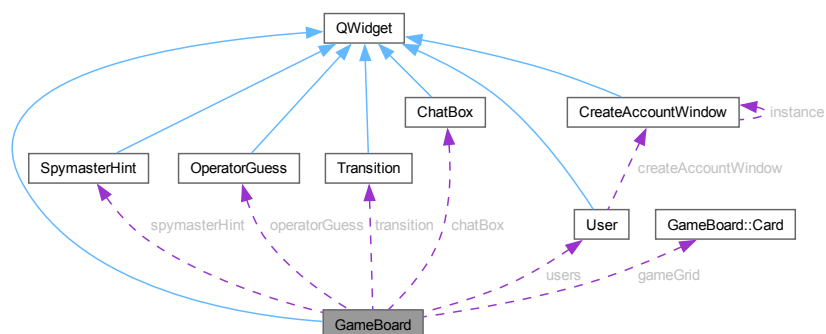
A class representing the game board for the Spy Master game.

```
#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.*

## 4.5.1 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

## 4.5.2 Member Enumeration Documentation

### 4.5.2.1 CardType

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

Enumeration for card types.

Enumerator

RED_TEAM	
BLUE_TEAM	
NEUTRAL	
ASSASSIN	

### 4.5.2.2 Turn

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

Enumeration representing the different turn states in the game board.

## Enumerator

RED_SPY	
RED_OP	
BLUE_SPY	
BLUE_OP	

### 4.5.3 Constructor & Destructor Documentation

#### 4.5.3.1 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

#### 4.5.3.2 ~GameBoard()

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

Destructor for the [GameBoard](#) class.

Cleans up resources used by the game board.

## Author

Group 9

## 4.5.4 Member Function Documentation

### 4.5.4.1 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

### 4.5.4.2 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

### 4.5.4.3 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

### 4.5.4.4 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

**4.5.4.5 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

**4.5.4.6 generateGameGrid()**

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

Generates the game grid.

Generates the game grid based on the loaded words.

## Author

Group 9

**4.5.4.7 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

#### 4.5.4.8 nextTurn()

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

Switches to the next turn.

Switches to the next turn and updates the UI.

Author

Group 9

#### 4.5.4.9 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

#### 4.5.4.10 onContinueClicked()

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

Handles the continue button click event.

Handles the continue button click event and updates the UI.

Author

Group 9

#### 4.5.4.11 resetGame()

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

Resets the game state.

Resets the game state to the initial state.

Author

Group 9

#### 4.5.4.12 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

**4.5.4.13 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

**4.5.4.14 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

**4.5.4.15 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

**4.5.4.16 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

**4.5.4.17 show**

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

Displays the game board.

Displays the game board and sets up the UI.

**Author**

Group 9

**4.5.4.18 showTransition()**

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

Displays a transition screen.

Displays a transition screen and updates the UI.

**Author**

Group 9

#### 4.5.4.19 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

#### 4.5.4.20 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

### 4.5.5 Member Data Documentation

#### 4.5.5.1 blueCardsRemaining

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

The number of remaining cards for each team.

#### 4.5.5.2 blueOperativeName

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

The names of the operative for the blue team.

#### 4.5.5.3 blueScoreLabel

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

The label for blue team score.

#### 4.5.5.4 blueSpyMasterName

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

The names of the spymaster for the blue team.

#### 4.5.5.5 blueTeamLabel

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

The label for blue team information.

#### 4.5.5.6 cards

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

The buttons representing the cards in the game grid.

#### 4.5.5.7 chatBox

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

The chat box widget.

#### 4.5.5.8 correspondingNumber

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

The label for the corresponding number.

#### 4.5.5.9 currentGuesses

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

The number of guesses made in the current turn.

#### 4.5.5.10 currentHint

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

The label for the current hint.

#### 4.5.5.11 currentPlayerName

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

The name of the current player.

#### 4.5.5.12 currentPlayerTeam

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

The team of the current player.

#### 4.5.5.13 currentTurn

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

Structure representing a turn in the game board.

#### 4.5.5.14 currentTurnLabel

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

The label for the current turn.

#### 4.5.5.15 gameGrid

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

The game grid.

#### 4.5.5.16 GRID\_SIZE

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

The size of the game grid.

#### 4.5.5.17 gridLayout

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

The grid layout for the game board.

#### 4.5.5.18 maxGuesses

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

The maximum number of guesses allowed in a turn.

#### 4.5.5.19 operatorGuess

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

The widget for the operator guess.

#### 4.5.5.20 redCardsRemaining

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

The number of remaining cards for each team.

#### 4.5.5.21 redOperativeName

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

The names of the operative for the blue team.

#### 4.5.5.22 redScoreLabel

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

The label for red team score.

#### 4.5.5.23 redSpyMasterName

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

The names of the spymaster for the red team.

#### 4.5.5.24 redTeamLabel

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

The labels for red team information.

#### 4.5.5.25 spymasterHint

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

The widget for the spymaster hint.

#### 4.5.5.26 transition

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

The transition screen widget.

#### 4.5.5.27 users

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

The list of users in the game.

#### 4.5.5.28 wordList

```
QStringList GameBoard::wordList [private]
```

The list of words.

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

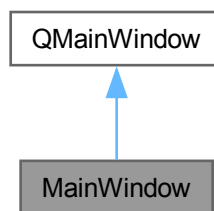
- [include/gameboard.h](#)
- [src/gameboard.cpp](#)

## 4.6 MainWindow Class Reference

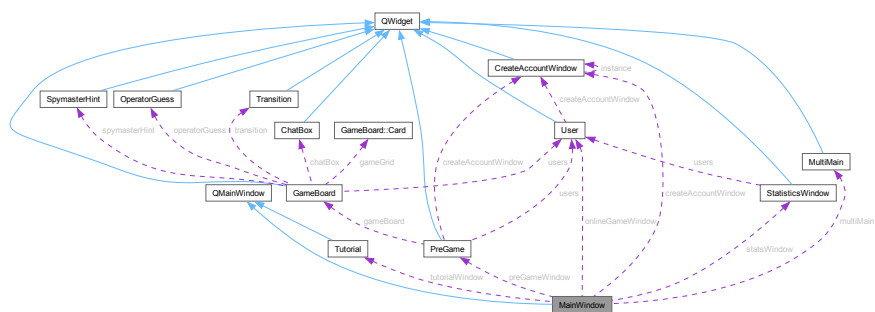
The main application window.

```
#include <mainwindow.h>
```

Inheritance diagram for MainWindow:



Collaboration diagram for MainWindow:



### 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 [openOnlineGame](#) ()  
*Opens the online game 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](#)
- [MultiMain](#) \* [multiMain](#)  
*Pointer to the Multiplayer main window.*
- 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.*
- [User](#) \* [onlineGameWindow](#)
- [CreateAccountWindow](#) \* [createAccountWindow](#)  
*Pointer to the account creation window.*
- [StatisticsWindow](#) \* [statsWindow](#)  
*Pointer to the statistics window displaying game stats.*
- [Tutorial](#) \* [tutorialWindow](#)



### 4.6.1 Detailed Description

The main application window.

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 MainWindow()

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

Constructor for [MainWindow](#).

Parameters

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

#### 4.6.2.2 ~MainWindow()

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

Destructor for [MainWindow](#).

### 4.6.3 Member Function Documentation

#### 4.6.3.1 openCreateAccount

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

Opens the Create Account window.

#### 4.6.3.2 openMultiMain

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

Opens the Multiplayer main window.

#### 4.6.3.3 openOnlineGame

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

Opens the online game window.

#### 4.6.3.4 openPreGame

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

Opens the [PreGame](#) window.

#### 4.6.3.5 openStatsWindow

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

Opens the statistics window.

#### 4.6.3.6 openTutorial

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

Opens the [Tutorial](#) window.

#### 4.6.3.7 showMainWindow

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

Displays the main window.

### 4.6.4 Member Data Documentation

#### 4.6.4.1 centralWidget

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

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

#### 4.6.4.2 createAccountButton

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

Button for opening the account creation window.

#### 4.6.4.3 createAccountWindow

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

Pointer to the account creation window.

#### 4.6.4.4 layout

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

Layout for organizing the widgets vertically.

#### 4.6.4.5 localPlayButton

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

Button for starting a local game.

#### 4.6.4.6 multiMain

```
MultiMain* MainWindow::multiMain [private]
```

Pointer to the Multiplayer main window.

#### 4.6.4.7 onlineGameWindow

```
User* MainWindow::onlineGameWindow [private]
```

Pointer to the online game window where users can play multiplayer.

#### 4.6.4.8 onlinePlayButton

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

Button for starting an online game.

#### 4.6.4.9 preGameWindow

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

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

#### 4.6.4.10 statsButton

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

Button for opening the statistics window.

#### 4.6.4.11 statsWindow

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

Pointer to the statistics window displaying game stats.

#### 4.6.4.12 titleLabel

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

Label displaying the application title.

#### 4.6.4.13 tutorialButton

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

Button for opening the tutorial.

#### 4.6.4.14 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:

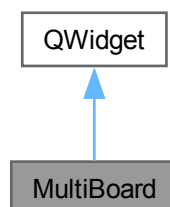
- include/[mainwindow.h](#)
- src/[mainwindow.cpp](#)

## 4.7 MultiBoard Class Reference

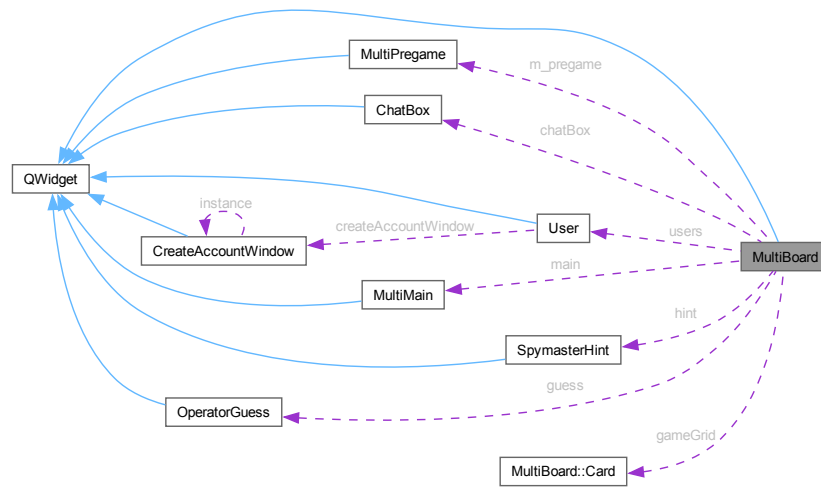
A widget that implements the multiplayer game board for a team-based word guessing game.

```
#include <multiboard.h>
```

Inheritance diagram for MultiBoard:



Collaboration diagram for MultiBoard:



## Classes

- struct [Card](#)  
Structure representing a card on the game board.

## Public Types

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

## Public Slots

- void [handleTileClick](#) ()  
Handles a player clicking on a tile in the game grid.
- void [processMessage](#) (const QString &message)  
Processes incoming network messages.
- void [socketDisconnected](#) ()  
Handles a disconnection event from the WebSocket.
- void [handleNewConnection](#) ()  
Handles a new client connection to the game server.

## Signals

- void [goBack](#) ()  
Signal emitted when returning to the previous screen.

## Public Member Functions

- [MultiBoard](#) (bool isHost, QWebSocketServer \*server, QList< QWebSocket \* > clients, QWebSocket \*clientSocket, const QHash< QString, QString > &playerRoles, const QString &currentUsername, QWidget \*parent=nullptr)

Constructor for the [MultiBoard](#) class.

## Private Member Functions

- void [setupUI](#) ()  
*Sets up the user interface for the game board.*
- void [setupBoard](#) ()  
*Sets up the game board with cards and initial state.*
- void [initializeWords](#) ()  
*Initializes the words for the game.*
- void [initializeBoardColors](#) ()  
*Initializes the board colors/teams for the game.*
- void [sendInitialGameState](#) ()  
*Sends the initial game state to all connected clients.*
- void [loadWordsFromFile](#) ()  
*Loads word list from a file.*
- void [generateGameGrid](#) ()  
*Generates the game grid layout.*
- void [checkGameEnd](#) ()  
*Checks if the game has ended.*
- void [processChatMessage](#) (const QString &playerName, const QString &message)  
*Processes a chat message from a player.*
- void [revealTile](#) (int row, int col, bool broadcast=true)  
*Reveals a tile on the game board.*
- void [advanceTurn](#) ()  
*Advances to the next turn in the game.*
- void [advanceTurnSpymaster](#) (const QString &hint, int number)  
*Advances the turn after a spymaster provides a hint.*
- void [updateTurnDisplay](#) ()  
*Updates the turn display for all players.*
- void [sendToAll](#) (const QString &message)  
*Sends a message to all connected players.*
- void [displayHint](#) (const QString &hint, int number)  
*Displays a hint to all players.*
- void [endGame](#) (const QString &message)  
*Ends the current game session.*
- bool [isMyTurn](#) () const  
*Checks if it's the current player's turn.*
- QString [getMyTeam](#) () const  
*Gets the team of the current player.*
- QString [getColorStyle](#) (const QString &color) const  
*Gets the CSS style for a specific card color.*

### Private Attributes

- bool [m\\_isHost](#)  
*Flag indicating if this instance is the host.*
- QWebSocketServer \* [m\\_server](#)  
*The WebSocket server for hosting.*
- QList< QWebSocket \* > [m\\_clients](#)  
*List of connected client sockets.*
- QWebSocket \* [m\\_clientSocket](#)  
*This player's client socket.*
- MultiPregame \* [m\\_pregame](#)  
*Reference to the pre-game setup screen.*
- ChatBox \* [chatBox](#)  
*Chat interface for player communication.*
- User \* [users](#)  
*User information management.*
- MultiMain \* [main](#)  
*Main game interface reference.*
- QHash< QString, QString > [m\\_playerRoles](#)  
*Mapping of usernames to roles.*
- QString [m\\_currentUsername](#)  
*Current player's username.*
- QString [m\\_currentRole](#)  
*Current player's role.*
- QVBoxLayout \* [gameVerticalLayout](#)  
*Vertical layout for the game.*
- QHBoxLayout \* [mainLayout](#)  
*Main horizontal layout.*
- QGridLayout \* [m\\_grid](#)  
*Grid layout for the game board.*
- QLabel \* [m\\_playerInfoLabel](#)  
*Label showing player information.*
- QLabel \* [m\\_turnLabel](#)  
*Label showing current turn.*
- QList< QPushButton \* > [m\\_tiles](#)  
*List of clickable word tiles.*
- SpymasterHint \* [hint](#)  
*Widget for spymaster to enter hints.*
- OperatorGuess \* [guess](#)  
*Widget for operators to make guesses.*
- QLabel \* [blueCardText](#)  
*Label showing blue cards remaining.*
- QLabel \* [redCardText](#)  
*Label showing red cards remaining.*
- QStringList [m\\_words](#)  
*List of words used in the game.*
- QStringList [m\\_tileColors](#)  
*List of card colors/teams.*
- QStringList [m\\_turnOrder](#)  
*Order of player turns.*
- int [m\\_currentTurnIndex](#)

- Index of the current turn.*
  - int [redCardsRemaining](#)
- Number of red team cards left.*
  - int [blueCardsRemaining](#)
- Number of blue team cards left.*
  - [Card](#) [gameGrid](#) [[GRID\\_SIZE](#)][[GRID\\_SIZE](#)]
- 2D array of game cards*
  - QStringList [wordList](#)
- List of available words.*
  - QPushButton \* [cards](#) [[GRID\\_SIZE](#)][[GRID\\_SIZE](#)]
- 2D array of card buttons*
  - QLabel \* [currentHint](#)
- Label showing current hint.*
  - QString [correspondingNumber](#)
- Number associated with current hint.*

### Static Private Attributes

- static const int [GRID\\_SIZE](#) = 5
- Size of the game grid (5x5)*

## 4.7.1 Detailed Description

A widget that implements the multiplayer game board for a team-based word guessing game.

The [MultiBoard](#) class manages the game state, UI, and network communications for a multiplayer word-guessing game. It handles player turns, card reveals, scoring, and game progression. The game involves two teams (red and blue) with spymasters giving hints and operators making guesses.

### Author

Group 9

## 4.7.2 Member Enumeration Documentation

### 4.7.2.1 CardType

enum [MultiBoard::CardType](#)

Enumeration of possible card types on the game board.

#### Enumerator

RED_TEAM	<a href="#">Card</a> belonging to the red team
BLUE_TEAM	<a href="#">Card</a> belonging to the blue team
NEUTRAL	Neutral card not belonging to either team
ASSASSIN	Assassin card that ends the game if selected

### 4.7.2.2 Turn

enum [MultiBoard::Turn](#)

Enumeration representing the different turn states in the game.



## Enumerator

RED_SPY	Red team spymaster's turn
RED_OP	Red team operator's turn
BLUE_SPY	Blue team spymaster's turn
BLUE_OP	Blue team operator's turn

### 4.7.3 Constructor & Destructor Documentation

#### 4.7.3.1 MultiBoard()

```
MultiBoard::MultiBoard (
    bool isHost,
    QWebSocketServer * server,
    QList< QWebSocket * > clients,
    QWebSocket * clientSocket,
    const QHash< QString, QString > & playerRoles,
    const QString & currentUsername,
    QWidget * parent = nullptr) [explicit]
```

Constructor for the [MultiBoard](#) class.

Initializes the game board with network connections and player information. Sets up the UI components and prepares the game state based on whether the player is hosting or joining a game.

## Parameters

<i>isHost</i>	Boolean indicating if this instance is the host of the game.
<i>server</i>	Pointer to the WebSocket server (if host).
<i>clients</i>	List of connected client WebSockets.
<i>clientSocket</i>	Pointer to this player's WebSocket (if not host).
<i>playerRoles</i>	Hash mapping player names to their roles.
<i>currentUsername</i>	The username of the current player.
<i>parent</i>	Optional parent widget.

## Author

Group 9

### 4.7.4 Member Function Documentation

#### 4.7.4.1 advanceTurn()

```
void MultiBoard::advanceTurn () [private]
```

Advances to the next turn in the game.

Updates the current turn state and notifies players of whose turn it is now.

## Author

Group 9

#### 4.7.4.2 advanceTurnSpymaster()

```
void MultiBoard::advanceTurnSpymaster (
    const QString & hint,
    int number) [private]
```

Advances the turn after a spymaster provides a hint.

Processes a spymaster's hint, displays it to all players, and changes the turn to the corresponding team's operator.

##### Parameters

<i>hint</i>	The word hint provided by the spymaster.
<i>number</i>	The number of cards the hint relates to.

##### Author

Group 9

#### 4.7.4.3 checkGameEnd()

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

Checks if the game has ended.

Evaluates the current game state to determine if either team has won or if the game should continue.

##### Author

Group 9

#### 4.7.4.4 displayHint()

```
void MultiBoard::displayHint (
    const QString & hint,
    int number) [private]
```

Displays a hint to all players.

Updates the UI to show the current hint and related number provided by a spymaster.

##### Parameters

<i>hint</i>	The word hint to display.
<i>number</i>	The number associated with the hint.

##### Author

Group 9

#### 4.7.4.5 endGame()

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

Ends the current game session.

Finalizes the game, shows the winning team, and prepares for a possible new game.

## Parameters

<i>message</i>	The end game message to display.
----------------	----------------------------------

## Author

Group 9

**4.7.4.6 generateGameGrid()**

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

Generates the game grid layout.

Creates the visual grid of cards with words and configures their initial appearance and behavior.

## Author

Group 9

**4.7.4.7 getColorStyle()**

```
QString MultiBoard::getColorStyle (
    const QString & color) const [private]
```

Gets the CSS style for a specific card color.

Returns the styling information for rendering cards of a particular team/color.

## Parameters

<i>color</i>	The color/team to get the style for.
--------------	--------------------------------------

## Returns

A string containing CSS style information.

## Author

Group 9

#### 4.7.4.8 getMyTeam()

```
QString MultiBoard::getMyTeam () const [private]
```

Gets the team of the current player.

Returns a string representing which team (red or blue) the current player belongs to.

##### Returns

A string containing the team name.

##### Author

Group 9

#### 4.7.4.9 goBack

```
void MultiBoard::goBack () [signal]
```

Signal emitted when returning to the previous screen.

Indicates that the player wants to leave the current game and return to the main menu or lobby.

##### Author

Group 9

#### 4.7.4.10 handleNewConnection

```
void MultiBoard::handleNewConnection () [slot]
```

Handles a new client connection to the game server.

Accepts new connections and sets up communication channels for new players joining the game. Only used when this instance is the host.

##### Author

Group 9

#### 4.7.4.11 handleTileClick

```
void MultiBoard::handleTileClick () [slot]
```

Handles a player clicking on a tile in the game grid.

Processes the action when a player clicks on a word tile, revealing the card's team affiliation if it's the player's turn to guess. Updates game state and advances turn if appropriate.

##### Author

Group 9

#### 4.7.4.12 initializeBoardColors()

```
void MultiBoard::initializeBoardColors () [private]
```

Initializes the board colors/teams for the game.

Assigns team affiliations (colors) to each card on the board, ensuring proper distribution of red, blue, neutral, and assassin cards.

**Author**

Group 9

#### 4.7.4.13 initializeWords()

```
void MultiBoard::initializeWords () [private]
```

Initializes the words for the game.

Loads or generates the set of words to be used for the current game session.

**Author**

Group 9

#### 4.7.4.14 isMyTurn()

```
bool MultiBoard::isMyTurn () const [private]
```

Checks if it's the current player's turn.

Determines whether the current player is allowed to perform actions based on the current turn state.

**Returns**

True if it's the current player's turn, false otherwise.

**Author**

Group 9

#### 4.7.4.15 loadWordsFromFile()

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

Loads word list from a file.

Reads the dictionary of possible words from a data file to use for populating the game board.

**Author**

Group 9

#### 4.7.4.16 processChatMessage()

```
void MultiBoard::processChatMessage (  
    const QString & playerName,  
    const QString & message) [private]
```

Processes a chat message from a player.

Handles incoming chat messages, displays them in the chat box, and checks for any game-related commands.

**Parameters**

<i>playerName</i>	The name of the player who sent the message.
<i>message</i>	The content of the chat message.

**Author**

Group 9

**4.7.4.17 processMessage**

```
void MultiBoard::processMessage (  
    const QString & message) [slot]
```

Processes incoming network messages.

Parses and handles various message types from other players, including game state updates, chat messages, and player actions.

**Parameters**

<i>message</i>	The message string received from the network.
----------------	---

**Author**

Group 9

**4.7.4.18 revealTile()**

```
void MultiBoard::revealTile (  
    int row,  
    int col,  
    bool broadcast = true) [private]
```

Reveals a tile on the game board.

Updates a card's state to revealed, shows its team affiliation, and updates the game state accordingly.

**Parameters**

<i>row</i>	The row of the tile in the grid.
<i>col</i>	The column of the tile in the grid.
<i>broadcast</i>	Whether to broadcast this action to other players.

**Author**

Group 9

#### 4.7.4.19 sendInitialState()

```
void MultiBoard::sendInitialState () [private]
```

Sends the initial game state to all connected clients.

Broadcasts the starting configuration of the game board to all players to ensure synchronization at game start.

Author

Group 9

#### 4.7.4.20 sendToAll()

```
void MultiBoard::sendToAll (
    const QString & message) [private]
```

Sends a message to all connected players.

Broadcasts a network message to all players in the game.

Parameters

<i>message</i>	The message to broadcast.
----------------	---------------------------

Author

Group 9

#### 4.7.4.21 setupBoard()

```
void MultiBoard::setupBoard () [private]
```

Sets up the game board with cards and initial state.

Initializes the game grid with words and card types, and configures the initial display state of all cards.

Author

Group 9

#### 4.7.4.22 setupUI()

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

Sets up the user interface for the game board.

Creates and arranges all UI components including the grid, information displays, and player controls.

Author

Group 9

#### 4.7.4.23 socketDisconnected

```
void MultiBoard::socketDisconnected () [slot]
```

Handles a disconnection event from the WebSocket.

Cleans up resources and updates the game state when a player disconnects from the game.

Author

Group 9

#### 4.7.4.24 updateTurnDisplay()

```
void MultiBoard::updateTurnDisplay () [private]
```

Updates the turn display for all players.

Refreshes the UI elements that show whose turn it is and what actions are available.

Author

Group 9

### 4.7.5 Member Data Documentation

#### 4.7.5.1 blueCardsRemaining

```
int MultiBoard::blueCardsRemaining [private]
```

Number of blue team cards left.

#### 4.7.5.2 blueCardText

```
QLabel* MultiBoard::blueCardText [private]
```

Label showing blue cards remaining.

#### 4.7.5.3 cards

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

2D array of card buttons

#### 4.7.5.4 chatBox

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

Chat interface for player communication.



#### 4.7.5.5 correspondingNumber

```
QString MultiBoard::correspondingNumber [private]
```

Number associated with current hint.

#### 4.7.5.6 currentHint

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

Label showing current hint.

#### 4.7.5.7 gameGrid

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

2D array of game cards

#### 4.7.5.8 gameVerticalLayout

```
QVBoxLayout* MultiBoard::gameVerticalLayout [private]
```

Vertical layout for the game.

#### 4.7.5.9 GRID\_SIZE

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

Size of the game grid (5x5)

#### 4.7.5.10 guess

```
OperatorGuess* MultiBoard::guess [private]
```

Widget for operators to make guesses.

#### 4.7.5.11 hint

```
SpymasterHint* MultiBoard::hint [private]
```

Widget for spymaster to enter hints.

#### 4.7.5.12 m\_clients

```
QList<QWebSocket*> MultiBoard::m_clients [private]
```

List of connected client sockets.

#### 4.7.5.13 m\_clientSocket

```
QWebSocket* MultiBoard::m_clientSocket [private]
```

This player's client socket.

#### 4.7.5.14 m\_currentRole

```
QString MultiBoard::m_currentRole [private]
```

Current player's role.

#### 4.7.5.15 m\_currentTurnIndex

```
int MultiBoard::m_currentTurnIndex [private]
```

Index of the current turn.

#### 4.7.5.16 m\_currentUsername

```
QString MultiBoard::m_currentUsername [private]
```

Current player's username.

#### 4.7.5.17 m\_grid

```
QGridLayout* MultiBoard::m_grid [private]
```

Grid layout for the game board.

#### 4.7.5.18 m\_isHost

```
bool MultiBoard::m_isHost [private]
```

Flag indicating if this instance is the host.

#### 4.7.5.19 m\_playerInfoLabel

```
QLabel* MultiBoard::m_playerInfoLabel [private]
```

Label showing player information.

#### 4.7.5.20 m\_playerRoles

```
QHash<QString, QString> MultiBoard::m_playerRoles [private]
```

Mapping of usernames to roles.

#### 4.7.5.21 m\_pregame

```
MultiPregame* MultiBoard::m_pregame [private]
```

Reference to the pre-game setup screen.

#### 4.7.5.22 m\_server

```
QWebSocketServer* MultiBoard::m_server [private]
```

The WebSocket server for hosting.

#### 4.7.5.23 m\_tileColors

```
QStringList MultiBoard::m_tileColors [private]
```

List of card colors/teams.

#### 4.7.5.24 m\_tiles

```
QList<QPushButton*> MultiBoard::m_tiles [private]
```

List of clickable word tiles.

#### 4.7.5.25 m\_turnLabel

```
QLabel* MultiBoard::m_turnLabel [private]
```

Label showing current turn.

#### 4.7.5.26 m\_turnOrder

```
QStringList MultiBoard::m_turnOrder [private]
```

Order of player turns.

#### 4.7.5.27 m\_words

```
QStringList MultiBoard::m_words [private]
```

List of words used in the game.

#### 4.7.5.28 main

```
MultiMain* MultiBoard::main [private]
```

Main game interface reference.

#### 4.7.5.29 mainLayout

```
QHBoxLayout* MultiBoard::mainLayout [private]
```

Main horizontal layout.

#### 4.7.5.30 redCardsRemaining

```
int MultiBoard::redCardsRemaining [private]
```

Number of red team cards left.

#### 4.7.5.31 redCardText

```
QLabel* MultiBoard::redCardText [private]
```

Label showing red cards remaining.

#### 4.7.5.32 users

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

[User](#) information management.

#### 4.7.5.33 wordList

```
QStringList MultiBoard::wordList [private]
```

List of available words.

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

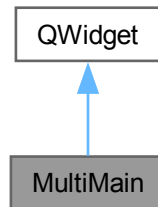
- include/Multiplayer/[multiboard.h](#)
- src/Multiplayer/[multiboard.cpp](#)

## 4.8 MultiMain Class Reference

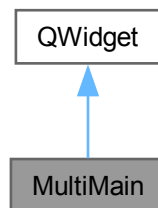
A widget that implements the main multiplayer lobby for creating and joining game rooms.

```
#include <multimain.h>
```

Inheritance diagram for MultiMain:



Collaboration diagram for MultiMain:



### Signals

- void [backToMainWindow](#) ()  
*Signal emitted when returning to the application's main window.*
- void [enterPregameAsHost](#) (QWebSocketServer \*server, const QString &username)  
*Signal emitted when entering the pre-game setup as a host.*
- void [enterPregameAsClient](#) (QWebSocket \*socket, const QString &username)  
*Signal emitted when entering the pre-game setup as a client.*

### Public Member Functions

- [MultiMain](#) (QWidget \*parent=nullptr)  
*Constructor for the [MultiMain](#) class.*
- [~MultiMain](#) ()  
*Destructor for the [MultiMain](#) class.*
- void [showMainWindow](#) ()  
*Displays the main lobby window.*

### Private Slots

- void [openMainWindow](#) ()  
*Opens the main application window.*
- void [onCreateRoomClicked](#) ()  
*Handles the user clicking the "Create Room" button.*
- void [onJoinRoomClicked](#) ()  
*Handles the user clicking the "Join Room" button.*
- void [onNewConnection](#) ()  
*Handles a new client connection to the game server.*
- void [processTextMessage](#) (QString message)  
*Processes an incoming text message from a WebSocket.*
- void [socketDisconnected](#) ()  
*Handles a WebSocket disconnection.*
- void [onConnected](#) ()  
*Handles successful connection to a host server.*
- void [onDisconnected](#) ()  
*Handles disconnection from a host server.*

### Private Member Functions

- void [updateLobbyList](#) ()  
*Updates the list of available game lobbies.*
- void [sendLobbyListToAll](#) ()  
*Sends the current lobby list to all connected clients.*

### Private Attributes

- QWebSocketServer \* [m\\_server](#) = nullptr  
*WebSocket server for hosting game rooms.*
- QWebSocket \* [m\\_clientSocket](#) = nullptr  
*Client WebSocket for joining rooms.*
- QList< QWebSocket \* > [m\\_clients](#)  
*List of connected client WebSockets.*
- QMap< QWebSocket \*, QString > [m\\_usernames](#)  
*Mapping of WebSockets to player usernames.*
- QString [m\\_username](#)  
*Current player's username.*
- QLabel \* [titleLabel](#)  
*Title label for the multiplayer lobby.*
- QPushButton \* [createRoomButton](#)  
*Button for creating a new game room.*
- QPushButton \* [joinRoomButton](#)  
*Button for joining an existing game room.*
- QPushButton \* [backButton](#)  
*Button for returning to the main window.*

### 4.8.1 Detailed Description

A widget that implements the main multiplayer lobby for creating and joining game rooms.

The [MultiMain](#) class provides the interface for players to either create a new game room as a host or join an existing game room as a client. It manages WebSocket connections for multiplayer functionality and handles the transition to the pre-game setup screen.

Author

Your Name

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 MultiMain()

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

Constructor for the [MultiMain](#) class.

Initializes the multiplayer lobby interface with buttons for creating and joining game rooms. Sets up the UI components and prepares network connections.

Parameters

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

Author

Your Name

#### 4.8.2.2 ~MultiMain()

```
MultiMain::~~MultiMain ()
```

Destructor for the [MultiMain](#) class.

Cleans up resources, including network connections and UI components.

Author

Your Name

### 4.8.3 Member Function Documentation

#### 4.8.3.1 backToMainWindow

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

Signal emitted when returning to the application's main window.

Indicates that the player wants to exit the multiplayer lobby and return to the main application window.

#### 4.8.3.2 enterPregameAsClient

```
void MultiMain::enterPregameAsClient (
    QWebSocket * socket,
    const QString & username) [signal]
```

Signal emitted when entering the pre-game setup as a client.

Triggered when a player joins an existing game room and transitions to the pre-game setup screen as a client.

**Parameters**

<i>socket</i>	Pointer to the client's WebSocket connection.
<i>username</i>	The username of the client player.

**4.8.3.3 enterPregameAsHost**

```
void MultiMain::enterPregameAsHost (  
    QWebSocketServer * server,  
    const QString & username) [signal]
```

Signal emitted when entering the pre-game setup as a host.

Triggered when a player creates a new game room and transitions to the pre-game setup screen as the host.

**Parameters**

<i>server</i>	Pointer to the WebSocket server instance.
<i>username</i>	The username of the host player.

**4.8.3.4 onConnected**

```
void MultiMain::onConnected () [private], [slot]
```

Handles successful connection to a host server.

Processes actions to take when a client successfully connects to a game room host.

**Author**

Your Name

**4.8.3.5 onCreateRoomClicked**

```
void MultiMain::onCreateRoomClicked () [private], [slot]
```

Handles the user clicking the "Create Room" button.

Creates a new game room with the current player as host, initializes the WebSocket server, and transitions to the pre-game setup.

**Author**

Your Name



#### 4.8.3.6 onDisconnected

```
void MultiMain::onDisconnected () [private], [slot]
```

Handles disconnection from a host server.

Processes actions to take when a client is disconnected from a game room host.

Author

Your Name

#### 4.8.3.7 onJoinRoomClicked

```
void MultiMain::onJoinRoomClicked () [private], [slot]
```

Handles the user clicking the "Join Room" button.

Connects to an existing game room as a client, establishes a WebSocket connection to the host, and transitions to the pre-game setup.

Author

Your Name

#### 4.8.3.8 onNewConnection

```
void MultiMain::onNewConnection () [private], [slot]
```

Handles a new client connection to the game server.

Accepts a new WebSocket connection from a client and sets up the communication channels.

Author

Your Name

#### 4.8.3.9 openMainWindow

```
void MultiMain::openMainWindow () [private], [slot]
```

Opens the main application window.

Handler for returning to the main application window from the multiplayer lobby.

Author

Your Name

#### 4.8.3.10 processTextMessage

```
void MultiMain::processTextMessage (  
    QString message) [private], [slot]
```

Processes an incoming text message from a WebSocket.

Handles and responds to various message types from connected clients or the host server.

**Parameters**

<i>message</i>	The text message received from the WebSocket.
----------------	---

**Author**

Your Name

**4.8.3.11 sendLobbyListToAll()**

```
void MultiMain::sendLobbyListToAll () [private]
```

Sends the current lobby list to all connected clients.

Broadcasts an updated list of available game rooms to all clients connected to this server.

**Author**

Your Name

**4.8.3.12 showMainWindow()**

```
void MultiMain::showMainWindow ()
```

Displays the main lobby window.

Makes the multiplayer lobby interface visible and sets up initial state.

**Author**

Your Name

**4.8.3.13 socketDisconnected**

```
void MultiMain::socketDisconnected () [private], [slot]
```

Handles a WebSocket disconnection.

Cleans up resources and updates the lobby state when a client disconnects from the server.

**Author**

Your Name

#### 4.8.3.14 updateLobbyList()

```
void MultiMain::updateLobbyList () [private]
```

Updates the list of available game lobbies.

Refreshes the UI with the current list of available game rooms that players can join.

Author

Your Name

### 4.8.4 Member Data Documentation

#### 4.8.4.1 backButton

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

Button for returning to the main window.

#### 4.8.4.2 createRoomButton

```
QPushButton* MultiMain::createRoomButton [private]
```

Button for creating a new game room.

#### 4.8.4.3 joinRoomButton

```
QPushButton* MultiMain::joinRoomButton [private]
```

Button for joining an existing game room.

#### 4.8.4.4 m\_clients

```
QList<QWebSocket*> MultiMain::m_clients [private]
```

List of connected client WebSockets.

#### 4.8.4.5 m\_clientSocket

```
QWebSocket* MultiMain::m_clientSocket = nullptr [private]
```

Client WebSocket for joining rooms.

#### 4.8.4.6 m\_server

```
QWebSocketServer* MultiMain::m_server = nullptr [private]
```

WebSocket server for hosting game rooms.

#### 4.8.4.7 m\_username

```
QString MultiMain::m_username [private]
```

Current player's username.

#### 4.8.4.8 m\_usernames

```
QMap<QWebSocket*, QString> MultiMain::m_usernames [private]
```

Mapping of WebSockets to player usernames.

#### 4.8.4.9 titleLabel

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

Title label for the multiplayer lobby.

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

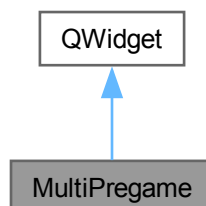
- [include/Multiplayer/multimain.h](#)
- [src/Multiplayer/multimain.cpp](#)

## 4.9 MultiPregame Class Reference

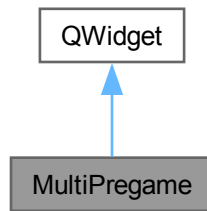
A widget that implements the pre-game lobby for multiplayer games.

```
#include <multipregame.h>
```

Inheritance diagram for MultiPregame:



Collaboration diagram for MultiPregame:



### Public Slots

- void [onNewConnection](#) ()  
*Handles a new client connection to the game server.*
- void [processMessage](#) (const QString &message)  
*Processes an incoming message from a WebSocket.*
- void [socketDisconnected](#) ()  
*Handles a WebSocket disconnection.*
- void [startGame](#) ()  
*Starts the game.*

### Signals

- void [backToMultiMain](#) ()  
*Signal emitted when returning to the multiplayer main menu.*
- void [enterPregameAsHost](#) (QWebSocketServer \*server, const QString &username)  
*Signal emitted when entering the pre-game setup as a host.*
- void [enterPregameAsClient](#) (QWebSocket \*socket, const QString &username)  
*Signal emitted when entering the pre-game setup as a client.*

### Public Member Functions

- [MultiPregame](#) (QWebSocketServer \*server, const QString &username, QWidget \*parent=nullptr)  
*Constructor for the [MultiPregame](#) class when acting as a host.*
- [MultiPregame](#) (QWebSocket \*socket, const QString &username, QWidget \*parent=nullptr)  
*Constructor for the [MultiPregame](#) class when acting as a client.*
- [~MultiPregame](#) ()  
*Destructor for the [MultiPregame](#) class.*
- void [clearUI](#) ()  
*Clears the user interface elements.*

## Private Member Functions

- void [resetUIState](#) ()  
*Resets the UI state to its initial condition.*
- void [setupUI](#) ()  
*Sets up the user interface for the pre-game lobby.*
- void [sendLobbyUpdate](#) ()  
*Sends an updated lobby state to all connected clients.*
- void [handleRoleSelection](#) (const QString &message, QWebSocket \*sender)  
*Handles a role selection message from a player.*
- void [gameStarted](#) (bool isHost, QWebSocketServer \*server, const QList< QWebSocket \* > &clients, QWebSocket \*clientSocket, const QHash< QString, QString > &playerRoles)  
*Handles the game start transition.*
- void [showPregame](#) ()  
*Displays the pre-game lobby interface.*

## Private Attributes

- QWebSocketServer \* [m\\_server](#) = nullptr  
*WebSocket server for hosting (nullptr for clients)*
- QWebSocket \* [m\\_clientSocket](#) = nullptr  
*Client WebSocket connection (nullptr for hosts)*
- QList< QWebSocket \* > [m\\_clients](#)  
*List of connected client WebSockets.*
- QMap< QWebSocket \*, QString > [m\\_usernames](#)  
*Mapping of WebSockets to player usernames.*
- QMap< QWebSocket \*, QString > [m\\_roles](#)  
*Mapping of WebSockets to player roles.*
- QMap< QWebSocket \*, bool > [m\\_checked](#)  
*Mapping of WebSockets to player readiness state.*
- QListWidget \* [playerList](#)  
*Widget displaying the list of connected players.*
- QString [m\\_username](#)  
*Current player's username.*
- bool [m\\_isHost](#)  
*Boolean indicating if this instance is the host.*

## 4.9.1 Detailed Description

A widget that implements the pre-game lobby for multiplayer games.

The [MultiPregame](#) class manages the pre-game setup phase where players join the lobby, select their roles, and prepare for the game to start. It handles both host and client functionality, manages player connections, role assignments, and transitions to the game.

### Author

Group 9

## 4.9.2 Constructor & Destructor Documentation

### 4.9.2.1 MultiPregame() [1/2]

```
MultiPregame::MultiPregame (  
    QWebSocketServer * server,  
    const QString & username,  
    QWidget * parent = nullptr)
```

Constructor for the [MultiPregame](#) class when acting as a host.

Initializes the pre-game lobby for a host player who is creating a new game. Sets up the server to accept client connections and manages the player list.

#### Parameters

<i>server</i>	Pointer to the WebSocket server for hosting the game.
<i>username</i>	The username of the host player.
<i>parent</i>	Optional parent widget.

#### Author

Group 9

### 4.9.2.2 MultiPregame() [2/2]

```
MultiPregame::MultiPregame (  
    QWebSocket * socket,  
    const QString & username,  
    QWidget * parent = nullptr)
```

Constructor for the [MultiPregame](#) class when acting as a client.

Initializes the pre-game lobby for a client player who is joining an existing game. Sets up the connection to the host server and prepares the player for role selection.

#### Parameters

<i>socket</i>	Pointer to the WebSocket connection to the host.
<i>username</i>	The username of the client player.
<i>parent</i>	Optional parent widget.

#### Author

Group 9

#### 4.9.2.3 ~MultiPregame()

`MultiPregame::~MultiPregame ()`

Destructor for the [MultiPregame](#) class.

Cleans up resources, including network connections and UI components.

Author

Group 9

### 4.9.3 Member Function Documentation

#### 4.9.3.1 backToMultiMain

`void MultiPregame::backToMultiMain () [signal]`

Signal emitted when returning to the multiplayer main menu.

Indicates that the player wants to exit the pre-game lobby and return to the multiplayer main menu.

#### 4.9.3.2 clearUI()

`void MultiPregame::clearUI ()`

Clears the user interface elements.

Removes all UI elements from the widget and prepares it for rebuilding or transition to another state.

Author

Group 9

#### 4.9.3.3 enterPregameAsClient

```
void MultiPregame::enterPregameAsClient (  
    QWebSocket * socket,  
    const QString & username) [signal]
```

Signal emitted when entering the pre-game setup as a client.

Used for transitioning to or refreshing the pre-game lobby with the current player as a client.

Parameters

<i>socket</i>	Pointer to the client's WebSocket connection.
<i>username</i>	The username of the client player.

#### 4.9.3.4 enterPregameAsHost

```
void MultiPregame::enterPregameAsHost (  
    QWebSocketServer * server,  
    const QString & username) [signal]
```

Signal emitted when entering the pre-game setup as a host.

Used for transitioning to or refreshing the pre-game lobby with the current player as the host.



## Parameters

<i>server</i>	Pointer to the WebSocket server instance.
<i>username</i>	The username of the host player.

**4.9.3.5 gameStarted()**

```
void MultiPregame::gameStarted (
    bool isHost,
    QWebSocketServer * server,
    const QList< QWebSocket * > & clients,
    QWebSocket * clientSocket,
    const QHash< QString, QString > & playerRoles) [private]
```

Handles the game start transition.

Sets up the necessary data and transitions to the game board when the game is started by the host.

## Parameters

<i>isHost</i>	Boolean indicating if this instance is the host.
<i>server</i>	Pointer to the WebSocket server (if host).
<i>clients</i>	List of connected client WebSockets.
<i>clientSocket</i>	Pointer to this player's WebSocket (if client).
<i>playerRoles</i>	Hash mapping player names to their selected roles.

## Author

Group 9

**4.9.3.6 handleRoleSelection()**

```
void MultiPregame::handleRoleSelection (
    const QString & message,
    QWebSocket * sender) [private]
```

Handles a role selection message from a player.

Processes a player's request to select a specific role, validates the selection, updates the lobby state, and notifies all clients.

## Parameters

<i>message</i>	The role selection message.
<i>sender</i>	Pointer to the WebSocket of the player making the selection.

## Author

Group 9

#### 4.9.3.7 onNewConnection

```
void MultiPregame::onNewConnection () [slot]
```

Handles a new client connection to the game server.

Accepts a new WebSocket connection from a client, adds them to the player list, and updates all connected clients. Only used when this instance is the host.

Author

Group 9

#### 4.9.3.8 processMessage

```
void MultiPregame::processMessage (
    const QString & message) [slot]
```

Processes an incoming message from a WebSocket.

Handles and responds to various message types from connected clients or the host server, including role selections and game start notifications.

Parameters

<i>message</i>	The message received from the WebSocket.
----------------	--

Author

Group 9

#### 4.9.3.9 resetUIState()

```
void MultiPregame::resetUIState () [private]
```

Resets the UI state to its initial condition.

Clears player selections, role assignments, and readiness states, preparing the UI for a fresh lobby state.

Author

Group 9

#### 4.9.3.10 sendLobbyUpdate()

```
void MultiPregame::sendLobbyUpdate () [private]
```

Sends an updated lobby state to all connected clients.

Broadcasts the current player list, role assignments, and readiness states to all connected clients to keep everyone synchronized.

Author

Group 9

#### 4.9.3.11 setupUI()

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

Sets up the user interface for the pre-game lobby.

Creates and arranges all UI components including the player list, role selection controls, and game start button.

Author

Group 9

#### 4.9.3.12 showPregame()

```
void MultiPregame::showPregame () [private]
```

Displays the pre-game lobby interface.

Shows the pre-game lobby UI and updates it with the current player list and role selections.

Author

Group 9

#### 4.9.3.13 socketDisconnected

```
void MultiPregame::socketDisconnected () [slot]
```

Handles a WebSocket disconnection.

Cleans up resources and updates the lobby state when a client disconnects from the server or the server disconnects.

Author

Group 9

#### 4.9.3.14 startGame

```
void MultiPregame::startGame () [slot]
```

Starts the game.

Initiates the game when all players are ready and roles are assigned. Sends game start notification to all clients and transitions to the game board. Only the host can trigger this action.

Author

Group 9

## 4.9.4 Member Data Documentation

### 4.9.4.1 m\_checked

```
QMap<QWebSocket*, bool> MultiPregame::m_checked [private]
```

Mapping of WebSockets to player readiness state.

### 4.9.4.2 m\_clients

```
QList<QWebSocket*> MultiPregame::m_clients [private]
```

List of connected client WebSockets.

### 4.9.4.3 m\_clientSocket

```
QWebSocket* MultiPregame::m_clientSocket = nullptr [private]
```

Client WebSocket connection (nullptr for hosts)

### 4.9.4.4 m\_isHost

```
bool MultiPregame::m_isHost [private]
```

Boolean indicating if this instance is the host.

### 4.9.4.5 m\_roles

```
QMap<QWebSocket*, QString> MultiPregame::m_roles [private]
```

Mapping of WebSockets to player roles.

### 4.9.4.6 m\_server

```
QWebSocketServer* MultiPregame::m_server = nullptr [private]
```

WebSocket server for hosting (nullptr for clients)

### 4.9.4.7 m\_username

```
QString MultiPregame::m_username [private]
```

Current player's username.

#### 4.9.4.8 m\_usernames

```
QMap<QWebSocket*, QString> MultiPregame::m_usernames [private]
```

Mapping of WebSockets to player usernames.

#### 4.9.4.9 playerList

```
QListWidget* MultiPregame::playerList [private]
```

Widget displaying the list of connected players.

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

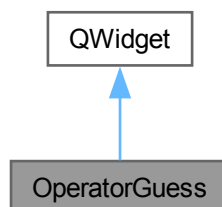
- [include/Multiplayer/multipregame.h](#)
- [src/Multiplayer/multipregame.cpp](#)

## 4.10 OperatorGuess Class Reference

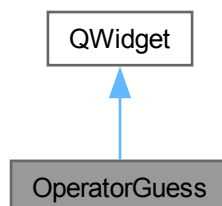
A widget that provides the interface for operators to submit guesses during gameplay.

```
#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.*

### 4.10.1 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

### 4.10.2 Constructor & Destructor Documentation

#### 4.10.2.1 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

#### 4.10.2.2 ~OperatorGuess()

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

Destructor for the [OperatorGuess](#) class.

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

## Author

Group 9

### 4.10.3 Member Function Documentation

#### 4.10.3.1 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.

#### 4.10.3.2 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

#### 4.10.3.3 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

#### 4.10.4 Member Data Documentation

##### 4.10.4.1 submitGuessButton

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

Button for submitting a guess.

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

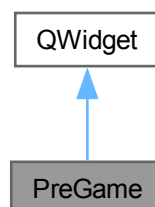
- [include/operatorguess.h](#)
- [src/operatorguess.cpp](#)

### 4.11 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.

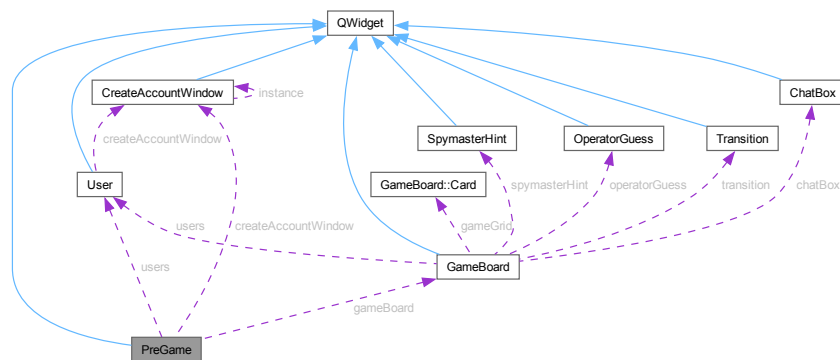
```
#include <pregame.h>
```

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.*

### 4.11.1 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.

### 4.11.2 Constructor & Destructor Documentation

#### 4.11.2.1 PreGame()

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

Construct a new Pre Game object.

##### Parameters

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

#### 4.11.2.2 ~PreGame()

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

Destroy the Pre Game object and clean up resources.

### 4.11.3 Member Function Documentation

#### 4.11.3.1 backToMainWindow

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

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

#### 4.11.3.2 getBlueTeamOperativeNickname()

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

Get the Blue Team Operative Nickname.

##### Returns

QString The nickname of the selected Blue Team Operative

#### 4.11.3.3 `getBlueTeamSpyMasterNickname()`

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

Get the Blue Team Spy Master Nickname.

##### Returns

QString The nickname of the selected Blue Team Spy Master

#### 4.11.3.4 `getRedTeamOperativeNickname()`

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

Get the Red Team Operative Nickname.

##### Returns

QString The nickname of the selected Red Team Operative

#### 4.11.3.5 `getRedTeamSpyMasterNickname()`

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

Get the Red Team Spy Master Nickname.

##### Returns

QString The nickname of the selected Red Team Spy Master

#### 4.11.3.6 `goBackToMain`

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

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

#### 4.11.3.7 `handleGameEnd`

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

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

#### 4.11.3.8 `openCreateAccount`

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

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

#### 4.11.3.9 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.

#### 4.11.3.10 show

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

Shows the pregame setup window and initializes user dropdowns.

#### 4.11.3.11 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.

#### 4.11.3.12 startGame

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

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

#### 4.11.3.13 update

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

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

### 4.11.4 Member Data Documentation

#### 4.11.4.1 backButton

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

Button to return to the main menu.

#### 4.11.4.2 blueTeamLayout

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

Vertical layout for the Blue Team's player selections.

#### 4.11.4.3 blueTeamOperativeComboBox

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

Dropdown menu for selecting the Blue Team's Operative.

#### 4.11.4.4 blueTeamSpyMasterComboBox

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

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

#### 4.11.4.5 buttonsLayout

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

Horizontal layout for the navigation buttons.

#### 4.11.4.6 createAccountButton

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

Button to open the account creation window.

#### 4.11.4.7 createAccountWindow

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

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

#### 4.11.4.8 gameBoard

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

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

#### 4.11.4.9 label

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

Title label for the pregame screen.

#### 4.11.4.10 layout

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

Main vertical layout for the entire pregame screen.

#### 4.11.4.11 redTeamLayout

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

Vertical layout for the Red Team's player selections.

#### 4.11.4.12 redTeamOperativeComboBox

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

Dropdown menu for selecting the Red Team's Operative.

#### 4.11.4.13 redTeamSpyMasterComboBox

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

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

#### 4.11.4.14 startButton

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

Button to start the game with selected players.

#### 4.11.4.15 teamsLayout

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

Horizontal layout to contain both team selection areas.

#### 4.11.4.16 usernames

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

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

#### 4.11.4.17 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:

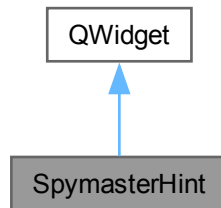
- [include/pregame.h](#)
- [src/pregame.cpp](#)

## 4.12 SpymasterHint Class Reference

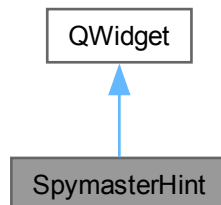
A widget for the spymaster to input a hint and the number of words associated with it.

```
#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.*

## 4.12.1 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

## 4.12.2 Constructor & Destructor Documentation

### 4.12.2.1 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.
---------------	--------------------

#### 4.12.2.2 ~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.

### 4.12.3 Member Function Documentation

#### 4.12.3.1 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.

#### 4.12.3.2 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.

#### 4.12.3.3 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.

#### 4.12.3.4 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.
-------------	-----------------

#### 4.12.3.5 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.

### 4.12.4 Member Data Documentation

#### 4.12.4.1 giveClueButton

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

QPushButton to submit the hint.

#### 4.12.4.2 hintLineEdit

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

QLineEdit used by the spymaster to input the hint.

#### 4.12.4.3 numberSpinBox

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

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

#### 4.12.4.4 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:

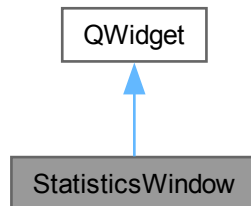
- [include/spymasterhint.h](#)
- [src/spymasterhint.cpp](#)

## 4.13 StatisticsWindow Class Reference

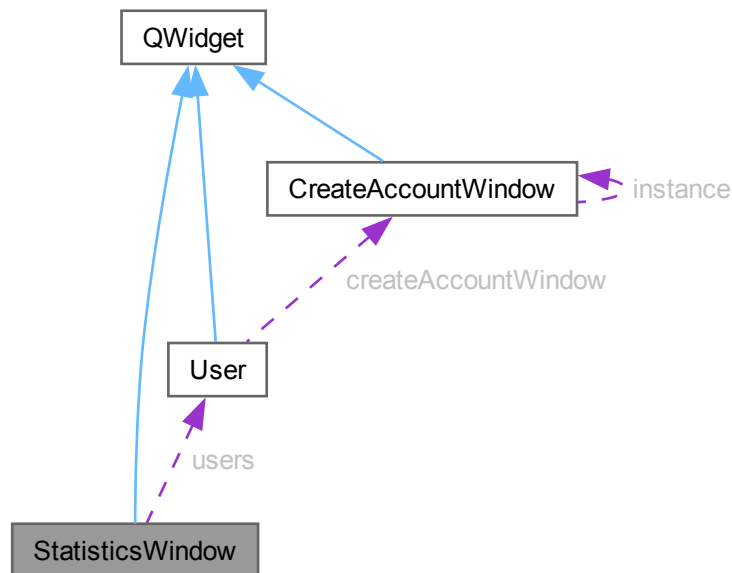
The class that shows the Statistics screen Displays game statistics for selected users including win rates and guess accuracy.

```
#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)*

### 4.13.1 Detailed Description

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

### 4.13.2 Constructor & Destructor Documentation

#### 4.13.2.1 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
---------------	---

#### 4.13.2.2 ~StatisticsWindow()

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

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

### 4.13.3 Member Function Documentation

#### 4.13.3.1 backToMainWindow

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

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

#### 4.13.3.2 goBackToMain

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

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

#### 4.13.3.3 populateDropDown()

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

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

#### 4.13.3.4 show

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

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

#### 4.13.3.5 showUserStats

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

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

### 4.13.4 Member Data Documentation

#### 4.13.4.1 backToMainButton

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

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

#### 4.13.4.2 gamesPlayedStats

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

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

#### 4.13.4.3 gamesWinRateStats

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

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

#### 4.13.4.4 gamesWinStats

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

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

#### 4.13.4.5 guessHitRateStats

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

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

#### 4.13.4.6 guessHitStats

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

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

#### 4.13.4.7 guessTotalStats

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

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

#### 4.13.4.8 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

#### 4.13.4.9 username

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

the username of the user Stores the currently selected username

#### 4.13.4.10 usernameComboBox

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

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

#### 4.13.4.11 usernameTitle

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

title of username Display label showing selected user's name

#### 4.13.4.12 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](#)

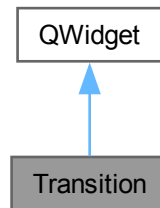


## 4.14 Transition Class Reference

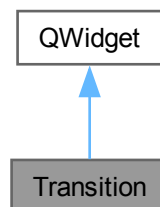
A widget for displaying a transition message and a button to continue.

```
#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.*

## 4.14.1 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

## 4.14.2 Constructor & Destructor Documentation

### 4.14.2.1 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.
---------------	--------------------

### 4.14.2.2 ~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.

## 4.14.3 Member Function Documentation

### 4.14.3.1 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.

### 4.14.3.2 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.
----------------	-------------------

#### 4.14.4 Member Data Documentation

##### 4.14.4.1 `continueButton`

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

The button that allows the user to continue.

##### 4.14.4.2 `messageLabel`

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

The label that displays the transition message.

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

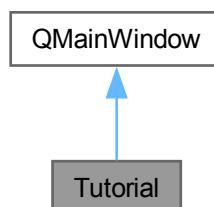
- [include/transition.h](#)
- [src/transition.cpp](#)

## 4.15 Tutorial Class Reference

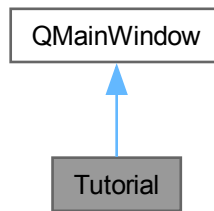
The tutorial window that guides users through the game mechanics.

```
#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.*

**4.15.1 Detailed Description**

The tutorial window that guides users through the game mechanics.

**4.15.2 Constructor & Destructor Documentation****4.15.2.1 Tutorial()**

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

Constructor for [Tutorial](#).

**Parameters**

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

**4.15.2.2 ~Tutorial()**

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

Destructor for [Tutorial](#).

**4.15.3 Member Function Documentation****4.15.3.1 closeEvent()**

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

Handles the close event.

#### Parameters

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

#### 4.15.3.2 onContinueClicked

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

Handles the continue button click event.

#### 4.15.3.3 resetTutorial()

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

Resets the tutorial to its initial state.

#### 4.15.3.4 tutorialClosed

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

Signal emitted when the tutorial is closed.

#### 4.15.3.5 updateContinueButtonPosition()

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

Updates the position of the continue button.

### 4.15.4 Member Data Documentation

#### 4.15.4.1 centralWidget

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

Pointer to the central widget.

#### 4.15.4.2 clickCount

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

Counter for continue button clicks.

#### 4.15.4.3 continueButton

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

Button for continuing through the tutorial.

#### 4.15.4.4 textBox

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

Label for displaying tutorial text.

#### 4.15.4.5 titleLabel

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

Label for the tutorial title.

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

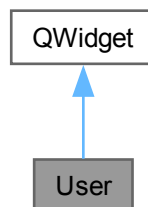
- [include/tutorial.h](#)
- [src/tutorial.cpp](#)

## 4.16 User Class Reference

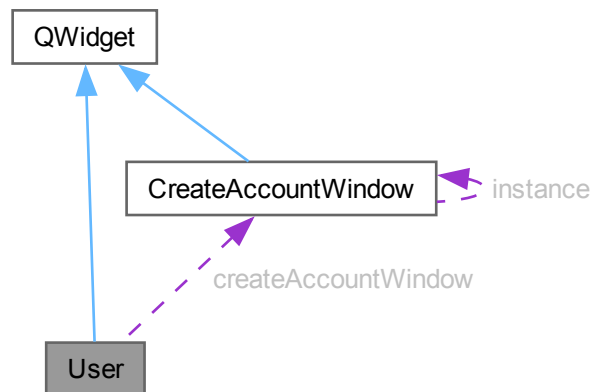
[User](#) class to handle local log in and loading/storing json files. This is a singleton class to ensure only one instance of user management exists. Manages user profiles, statistics, and authentication.

```
#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*

## 4.16.1 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.

## 4.16.2 Constructor & Destructor Documentation

### 4.16.2.1 ~User()

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

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

### 4.16.2.2 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
---------------	--

## 4.16.3 Member Function Documentation

### 4.16.3.1 backToMainMenu

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

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

### 4.16.3.2 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

**4.16.3.3 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

**4.16.3.4 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

**4.16.3.5 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)

**4.16.3.6 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)

**4.16.3.7 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

**4.16.3.8 handleCreateAccount**

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

create user account Opens account creation window

**4.16.3.9 handleLogin**

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

log in the user Handles authentication and session creation

**4.16.3.10 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
-----------------	----------------------

**4.16.3.11 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

**4.16.3.12 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

**4.16.3.13 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
-----------------	----------------------

**4.16.3.14 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
-----------------	----------------------

**4.16.3.15 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
-------------------	-----------------------

**4.16.3.16 refreshUserDropdown**

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

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

**4.16.3.17 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

**4.16.3.18 show**

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

show the current screen Makes the user login UI visible

**4.16.3.19 showMainMenu**

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

show the main menu Returns to main application screen

**4.16.3.20 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

**4.16.3.21 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

**4.16.3.22 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

**4.16.3.23 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

**4.16.3.24 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

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

## 4.16.4 Member Data Documentation

### 4.16.4.1 backButton

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

the button to go back UI element for navigation

### 4.16.4.2 createAccountButton

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

the button to create account UI element to open account creation

### 4.16.4.3 createAccountWindow

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

variable that stores the create account window Manages account creation UI

### 4.16.4.4 jsonContentLabel

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

the text to show the debug UI element for displaying information

### 4.16.4.5 jsonFilePath

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

the path of the users info Location of JSON profile storage

### 4.16.4.6 loginButton

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

the button to log in UI element for authentication

### 4.16.4.7 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

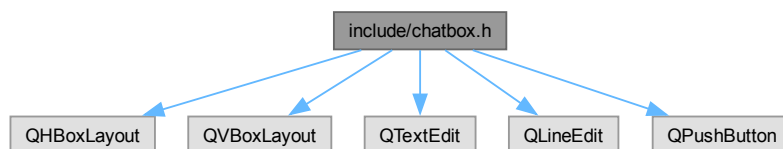
# File Documentation

### 5.1 include/chatbox.h File Reference

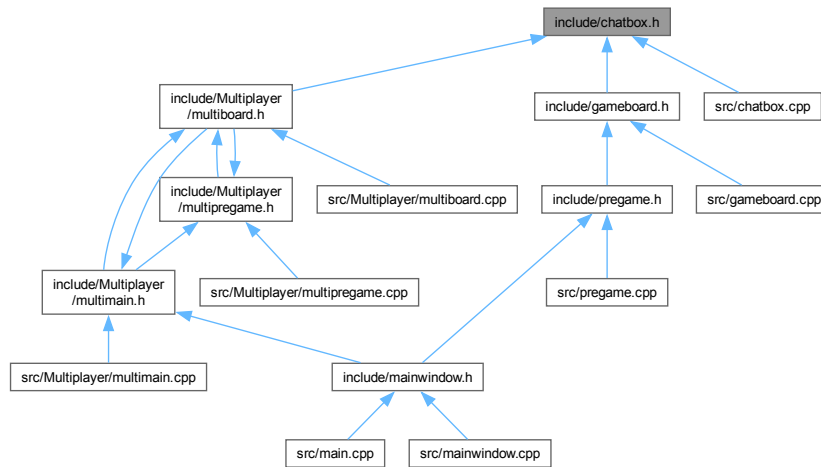
Header file for the [ChatBox](#) class, which provides a UI for the chat feature in the game.

```
#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.

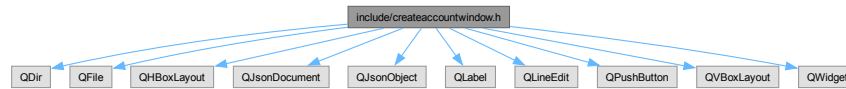
```

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

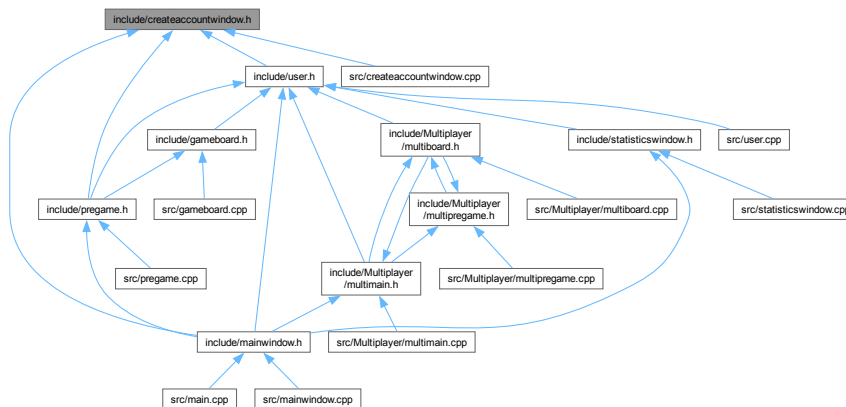
```

```
#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 "Multiplayer/multimain.h"
#include "createaccountwindow.h"
#include "pregame.h"
#include "statisticswindow.h"
#include "tutorial.h"
#include "user.h"

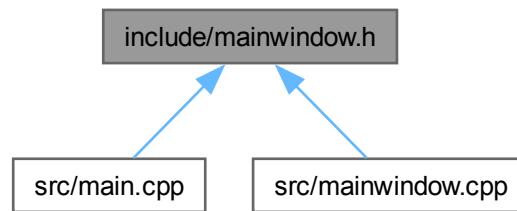
```

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
00002
00003 #ifndef MAINWINDOW_H
00004 #define MAINWINDOW_H
00005
00006 #include <QGraphicsDropShadowEffect>
00007 #include <QGuiApplication>
00008 #include <QLabel>
00009 #include <QMainWindow>
00010 #include <QPalette>
00011 #include <QPixmap>
00012 #include <QPushButton>
00013 #include <QScreen>
00014 #include <QVBoxLayout>
00015
00016 #include "Multiplayer/multimain.h"
00017 #include "createaccountwindow.h"
00018 #include "pregame.h"
00019 #include "statisticswindow.h"
00020 #include "tutorial.h"
00021 #include "user.h"
00022
00023 class PreGame;
00024 class User;
00025 class CreateAccountWindow;
00026 class StatisticsWindow;
00027 class Tutorial;
00028 class MultiMain;
00029
00030 class MainWindow : public QMainWindow {
00031     Q_OBJECT
00032
00033     public:
00034         explicit MainWindow(QWidget* parent = nullptr);
00035
00036
00037
00038
00039
00040
00041
00042
00043
00044
00045
00046
  
```

```

00050 ~MainWindow();
00051
00052 public slots:
00056 void showMainWindow();
00057
00058 private slots:
00062 void openPreGame();
00063
00067 void openOnlineGame();
00068
00072 void openStatsWindow();
00073
00077 void openCreateAccount();
00078
00082 void openTutorial();
00083
00087 void openMultiMain();
00088
00089 private:
00090 QWidget* centralWidget;
00092 QVBoxLayout* layout;
00093
00094 QLabel* titleLabel;
00095
00096 PreGame* preGameWindow;
00098 MultiMain* multiMain;
00099 QPushButton* localPlayButton;
00100 QPushButton* onlinePlayButton;
00101 QPushButton* tutorialButton;
00102 QPushButton* statsButton;
00103 QPushButton*
00104     createAccountButton;
00105
00106 User* onlineGameWindow;
00108 CreateAccountWindow*
00109     createAccountWindow;
00110 StatisticsWindow*
00111     statsWindow;
00112 Tutorial* tutorialWindow;
00114 };
00115
00116 #endif // MAINWINDOW_H

```

## 5.9 include/Multiplayer/multiboard.h File Reference

Header file for the [MultiBoard](#) class, which implements a multiplayer game board.

```

#include <QDebug>
#include <QFile>
#include <QGridLayout>
#include <QHBoxLayout>
#include <QHash>
#include <QLabel>
#include <QList>
#include <QMessageBox>
#include <QPushButton>
#include <QRandomGenerator>
#include <QString>
#include <QStringList>
#include <QVBoxLayout>
#include <QWebSocket>
#include <QWebSocketServer>
#include <QWidget>
#include "../operatorguess.h"
#include "../spymasterhint.h"
#include "Multiplayer/multimain.h"
#include "Multiplayer/multipregame.h"
#include "chatbox.h"

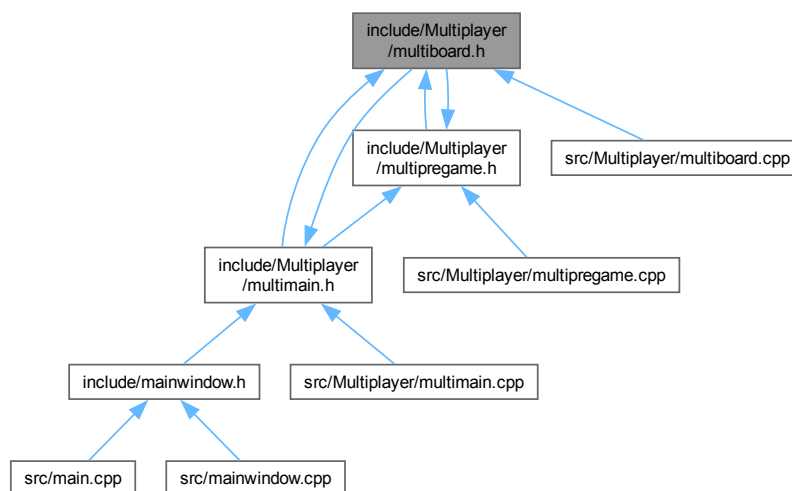
```

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

Include dependency graph for multiboard.h:



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



## Classes

- class [MultiBoard](#)  
A widget that implements the multiplayer game board for a team-based word guessing game.
- struct [MultiBoard::Card](#)  
Structure representing a card on the game board.

### 5.9.1 Detailed Description

Header file for the [MultiBoard](#) class, which implements a multiplayer game board.

Author

Group 9

## 5.10 multiboard.h

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

```

00001
00007
00008 #ifndef MULTIBOARD_H
00009 #define MULTIBOARD_H
00010
00011 #include <QDebug>
00012 #include <QFile>
00013 #include <QGridLayout>
00014 #include <QHBoxLayout>
00015 #include <QHash>
00016 #include <QLabel>
00017 #include <QList>
00018 #include <QMessageBox>
00019 #include <QPushButton>
00020 #include <QRandomGenerator>
00021 #include <QString>
00022 #include <QStringList>
00023 #include <QVBoxLayout>
00024 #include <QWebSocket>
00025 #include <QWebSocketServer>
00026 #include <QWidget>
00027
00028 #include "../operatorguess.h"
00029 #include "../spymasterhint.h"
00030 #include "Multiplayer/multimain.h"
00031 #include "Multiplayer/multipregame.h"
00032 #include "chatbox.h"
00033 #include "user.h"
00034
00035 class MultiMain;
00036 class MultiPregame;
00037
00050 class MultiBoard : public QWidget {
00051     Q_OBJECT
00052
00053 public:
00071     explicit MultiBoard(bool isHost, QWebSocketServer* server,
00072                         QList<QWebSocket*> clients, QWebSocket* clientSocket,
00073                         const QHash<QString, QString>& playerRoles,
00074                         const QString& currentUsername,
00075                         QWidget* parent = nullptr //,
00076                         // MultiPregame* pregame = nullptr
00077     );
00078
00083     enum CardType {
00084         RED_TEAM,
00085         BLUE_TEAM,
00086         NEUTRAL,
00087         ASSASSIN
00088     };
00089
00094     enum Turn {
00095         RED_SPY,
00096         RED_OP,
00097         BLUE_SPY,
00098         BLUE_OP
00099     };
00100
00105     struct Card {
00106         QString word;
00107         CardType type;
00108         bool revealed;
00109     };
00110
00111 public slots:
00121     void handleTileClick();
00122
00133     void processMessage(const QString& message);
00134
00143     void socketDisconnected();
00144
00153     void handleNewConnection();
00154
00155 signals:
00164     void goBack();
00165
00166 private:
00167     // Network setup
00168
00170     bool m_isHost;
00172     QWebSocketServer* m_server;
00174     QList<QWebSocket*> m_clients;

```

```

00176 QWebSocket* m_clientSocket;
00178 MultiPregame* m_pregame;
00179
00181 ChatBox* chatBox;
00182
00184 User* users;
00185
00187 MultiMain* main;
00188
00189 // Player information
00191 QHash<QString, QString> m_playerRoles;
00193 QString m_currentUsername;
00195 QString m_currentRole;
00196
00197 // Game board components
00199 QVBoxLayout* gameVerticalLayout;
00201 QHBoxLayout* mainLayout;
00203 QGridLayout* m_grid;
00205 QLabel* m_playerInfoLabel;
00207 QLabel* m_turnLabel;
00209 QList<QPushButton*> m_tiles;
00211 SpymasterHint* hint;
00213 OperatorGuess* guess;
00214
00216 QLabel* blueCardText;
00218 QLabel* redCardText;
00219
00220 // Game state
00222 QStringList m_words;
00224 QStringList m_tileColors;
00226 QStringList m_turnOrder;
00228 int m_currentTurnIndex;
00229
00230 // Cards remaining
00232 int redCardsRemaining;
00234 int blueCardsRemaining;
00235
00244 void setupUI();
00245
00254 void setupBoard();
00255
00264 void initializeWords();
00265
00274 void initializeBoardColors();
00275
00284 void sendInitialGameState();
00285
00294 void loadWordsFromFile();
00295
00304 void generateGameGrid();
00305
00314 void checkGameEnd();
00315
00327 void processChatMessage(const QString& playerName, const QString& message);
00328
00341 void revealTile(int row, int col, bool broadcast = true);
00342
00351 void advanceTurn();
00352
00364 void advanceTurnSpymaster(const QString& hint, int number);
00365
00374 void updateTurnDisplay();
00375
00385 void sendToAll(const QString& message);
00386
00398 void displayHint(const QString& hint, int number);
00399
00410 void endGame(const QString& message);
00411
00413 static const int GRID_SIZE = 5;
00415 Card gameGrid[GRID_SIZE][GRID_SIZE];
00417 QStringList wordList;
00419 QPushButton* cards[GRID_SIZE][GRID_SIZE];
00421 QLabel* currentHint;
00423 QString correspondingNumber;
00424
00435 bool isMyTurn() const;
00436
00447 QString getMyTeam() const;
00448
00460 QString getColorStyle(const QString& color) const;
00461 };
00462
00463 #endif // MULTIBOARD_H

```

## 5.11 include/Multiplayer/multimain.h File Reference

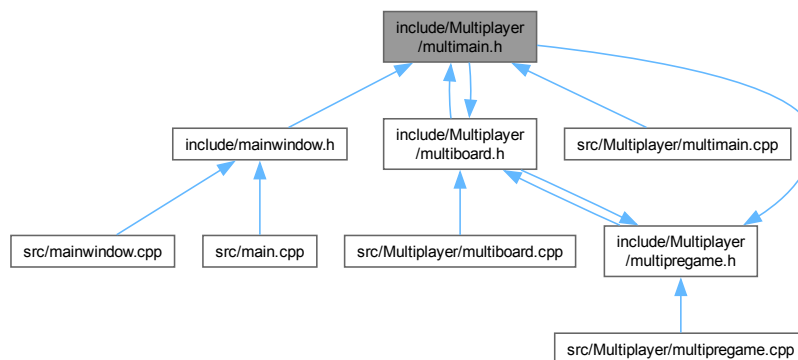
Header file for the [MultiMain](#) class, which implements the main multiplayer game lobby interface.

```
#include <QComboBox>
#include <QGraphicsDropShadowEffect>
#include <QGuiApplication>
#include <QInputDialog>
#include <QLabel>
#include <QListWidget>
#include <QMap>
#include <QMessageBox>
#include <QNetworkInterface>
#include <QPalette>
#include <QPixmap>
#include <QPushButton>
#include <QScreen>
#include <QVBoxLayout>
#include <QWebSocket>
#include <QWebSocketServer>
#include <QWidget>
#include "Multiplayer/multiboard.h"
#include "Multiplayer/multipregame.h"
#include "user.h"
```

Include dependency graph for multimain.h:



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



### Classes

- class [MultiMain](#)

*A widget that implements the main multiplayer lobby for creating and joining game rooms.*

### 5.11.1 Detailed Description

Header file for the [MultiMain](#) class, which implements the main multiplayer game lobby interface.

Author

Your Name

## 5.12 multimain.h

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

```

00001
00007
00008 #ifndef MULTIMAIN_H
00009 #define MULTIMAIN_H
00010
00011 #include <QComboBox>
00012 #include <QGraphicsDropShadowEffect>
00013 #include <QGuiApplication>
00014 #include <QInputDialog>
00015 #include <QLabel>
00016 #include <QListWidget>
00017 #include <QMap>
00018 #include <QMessageBox>
00019 #include <QNetworkInterface>
00020 #include <QPalette>
00021 #include <QPixmap>
00022 #include <QPushButton>
00023 #include <QScreen>
00024 #include <QVBoxLayout>
00025 #include <QWebSocket>
00026 #include <QWebSocketServer>
00027 #include <QWidget>
00028
00029 #include "Multiplayer/multiboard.h"
00030 #include "Multiplayer/multipregame.h"
00031 #include "user.h"
00032
00045 class MultiMain : public QWidget {
00046     Q_OBJECT
00047 public:
00059     explicit MultiMain(QWidget* parent = nullptr);
00060
00069     ~MultiMain();
00070
00079     void showMainWindow();
00080
00081 signals:
00088     void backToMainWindow();
00089
00099     void enterPregameAsHost(QWebSocketServer* server, const QString& username);
00100
00110     void enterPregameAsClient(QWebSocket* socket, const QString& username);
00111
00112 private slots:
00121     void openMainWindow();
00122
00131     void onCreateRoomClicked();
00132
00142     void onJoinRoomClicked();
00143
00152     void onNewConnection();
00153
00164     void processTextMessage(QString message);
00165
00174     void socketDisconnected();
00175
00184     void onConnected();
00185
00194     void onDisconnected();
00195
00196 private:
00197     // Network members
00199     QWebSocketServer* m_server = nullptr;
00201     QWebSocket* m_clientSocket = nullptr;
00203     QList<QWebSocket*> m_clients;
00205     QMap<QWebSocket*, QString> m_usernames;

```

```

00207   QString m_username;
00208
00209   // UI members
00211   QLabel* titleLabel;
00213   QPushButton* createRoomButton;
00215   QPushButton* joinRoomButton;
00217   QPushButton* backButton;
00218
00227   void updateLobbyList();
00228
00237   void sendLobbyListToAll();
00238 };
00239
00240 #endif // MULTIMAIN_H

```

## 5.13 include/Multiplayer/multipregame.h File Reference

Header file for the [MultiPregame](#) class, which implements the pre-game lobby for multiplayer games.

```

#include <QInputDialog>
#include <QLabel>
#include <QListWidget>
#include <QLoggingCategory>
#include <QMessageBox>
#include <QNetworkInterface>
#include <QPushButton>
#include <QVBoxLayout>
#include <QWebSocket>
#include <QWebSocketServer>
#include <QWidget>
#include "Multiplayer/multiboard.h"

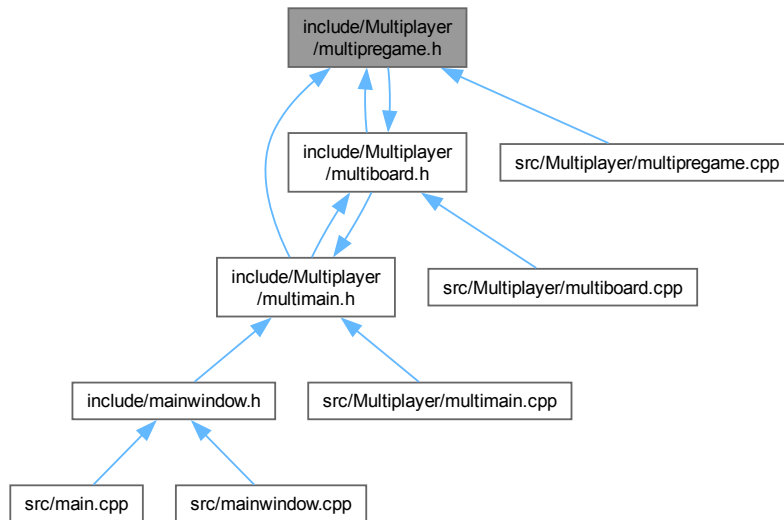
```

Include dependency graph for multipregame.h:





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



## Classes

- class [MultiPregame](#)

*A widget that implements the pre-game lobby for multiplayer games.*

### 5.13.1 Detailed Description

Header file for the [MultiPregame](#) class, which implements the pre-game lobby for multiplayer games.

Author

Group 9

## 5.14 multipregame.h

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

```

00001
00007
00008 #ifndef MULTIPREGAME_H
00009 #define MULTIPREGAME_H
00010
00011 #include <QInputDialog>
00012 #include <QLabel>
00013 #include <QListWidget>
00014 #include <QLoggingCategory>
00015 #include <QMessageBox>
00016 #include <QNetworkInterface>
00017 #include <QPushButton>
00018 #include <QVBoxLayout>
00019 #include <QWebSocket>
00020 #include <QWebSocketServer>
00021 #include <QWidget>
00022

```

```

00023 #include "Multiplayer/multiboard.h"
00024
00036 class MultiPregame : public QWidget {
00037     Q_OBJECT
00038 public:
00052     MultiPregame(QWebSocketServer* server, const QString& username,
00053                 QWidget* parent = nullptr);
00054
00068     MultiPregame(QWebSocket* socket, const QString& username,
00069                 QWidget* parent = nullptr);
00070
00079     ~MultiPregame();
00080
00089     void clearUI();
00090
00091 signals:
00098     void backToMultiMain();
00099
00109     void enterPregameAsHost(QWebSocketServer* server, const QString& username);
00110
00120     void enterPregameAsClient(QWebSocket* socket, const QString& username);
00121
00122 public slots:
00132     void onNewConnection();
00133
00145     void processMessage(const QString& message);
00146
00155     void socketDisconnected();
00156
00166     void startGame();
00167
00168 private:
00177     void resetUIState();
00178
00187     void setupUI();
00188
00197     void sendLobbyUpdate();
00198
00210     void handleRoleSelection(const QString& message, QWebSocket* sender);
00211
00226     void gameStarted(bool isHost, QWebSocketServer* server,
00227                      const QList<QWebSocket*>& clients, QWebSocket* clientSocket,
00228                      const QHash<QString, QString>& playerRoles);
00229
00238     void showPregame();
00239
00241     QWebSocketServer* m_server = nullptr;
00243     QWebSocket* m_clientSocket = nullptr;
00245     QList<QWebSocket*> m_clients;
00247     QMap<QWebSocket*, QString> m_usernames;
00249     QMap<QWebSocket*, QString> m_roles;
00251     QMap<QWebSocket*, bool> m_checked;
00252
00254     QListWidget* playerList;
00256     QString m_username;
00258     bool m_isHost;
00259 };
00260
00261 #endif // MULTIPREGAME_H

```

## 5.15 include/operatorguess.h File Reference

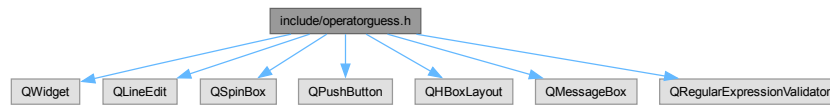
Header file for the [OperatorGuess](#) class, which handles operator guessing interface.

```

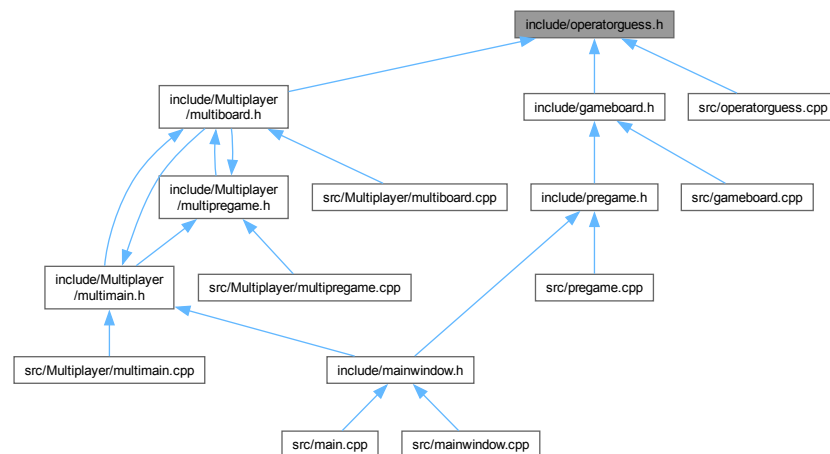
#include <QWidget>
#include <QLineEdit>
#include <QSpinBox>
#include <QPushButton>
#include <QHBoxLayout>
#include <QMessageBox>
#include <QRegularExpressionValidator>

```

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.*

### 5.15.1 Detailed Description

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

Author

Group 9

## 5.16 operatorguess.h

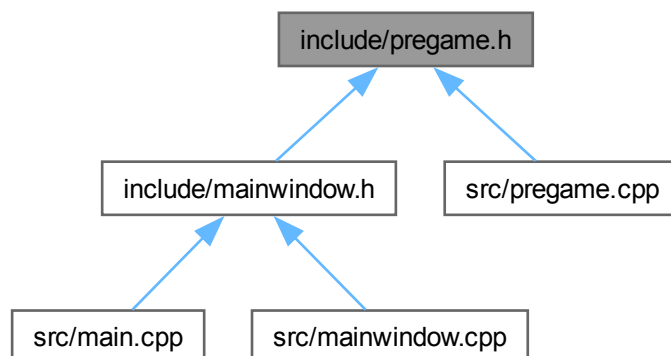
[Go to the documentation of this file.](#)

```

00001
00006
00007 #ifndef OPERATORGUESS_H
00008 #define OPERATORGUESS_H
00009
  
```



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.*

### 5.17.1 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

## 5.18 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

```

## 5.19 include/spymasterhint.h File Reference

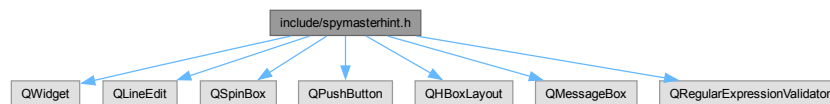
Header file for the [SpymasterHint](#) class, which provides a UI for the spymaster to give hints.

```

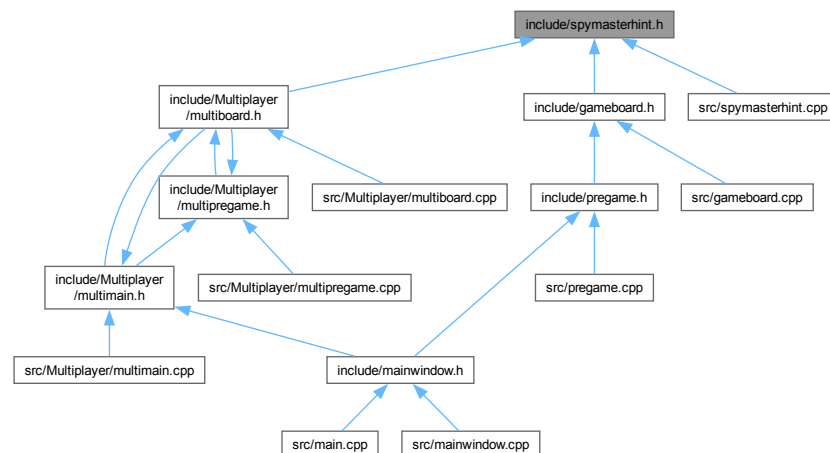
#include <QWidget>
#include <QLineEdit>
#include <QSpinBox>
#include <QPushButton>
#include <QHBoxLayout>
#include <QMessageBox>
#include <QRegularExpressionValidator>

```

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.*

### 5.19.1 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

## 5.20 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
```

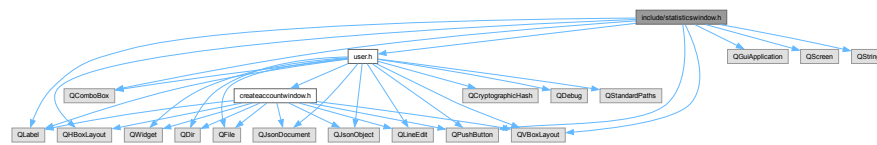


## 5.21 include/statisticswindow.h File Reference

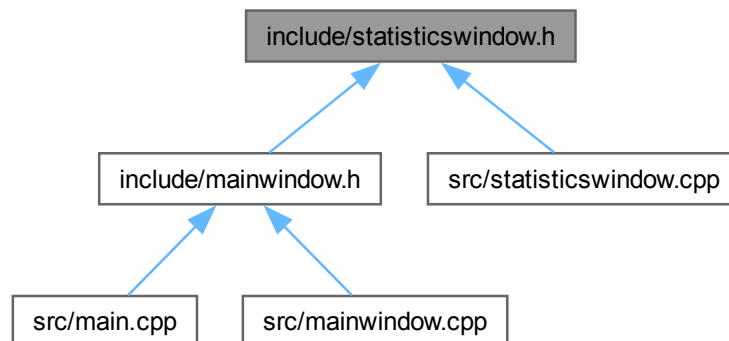
The screen to show the user's statistics.

```
#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.*

### 5.21.1 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

## 5.22 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

```

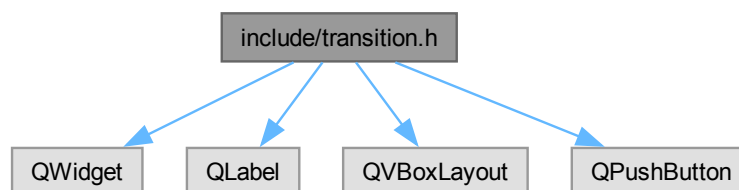
```
00127     QLabel* guessTotalStats;
00128
00133     QLabel* guessHitStats;
00134
00139     QLabel* guessHitRateStats;
00140
00141 private:
00146     void populateDropDown();
00147
00148 private slots:
00153     void goBackToMain();
00154
00159     void showUserStats();
00160 };
00161
00162 #endif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

## 5.23 include/transition.h File Reference

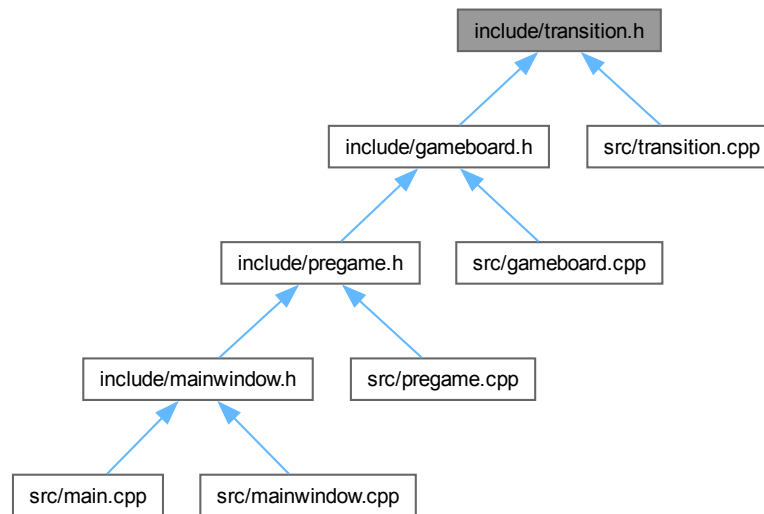
Header file for the [Transition](#) class, which provides a UI for transitions between game states.

```
#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.*

### 5.23.1 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

## 5.24 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

```

## 5.25 include/tutorial.h File Reference

Declaration of the [Tutorial](#) class.

```

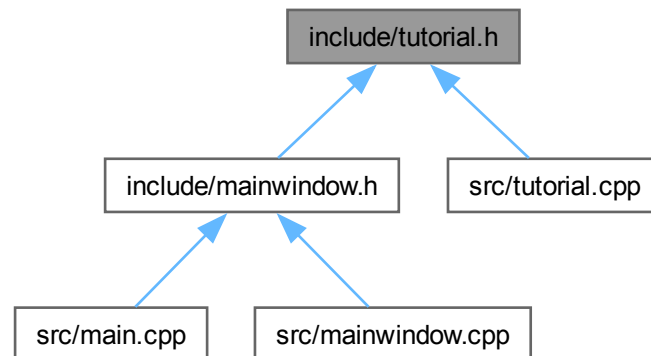
#include <QCloseEvent>
#include <QDebug>
#include <QDir>
#include <QFile>
#include <QGraphicsDropShadowEffect>
#include <QGuiApplication>
#include <QLabel>
#include <QMainWindow>
#include <QPushButton>
#include <QScreen>
#include <QVBoxLayout>
#include <QWidget>

```

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.*

### 5.25.1 Detailed Description

Declaration of the [Tutorial](#) class.

## 5.26 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

```

## 5.27 include/user.h File Reference

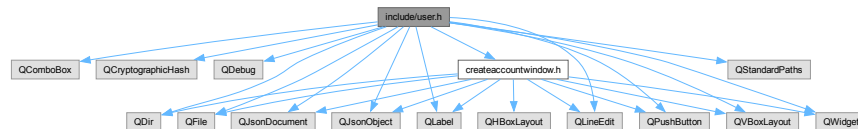
[User](#) class to handle local log in and loading/storing json files.

```

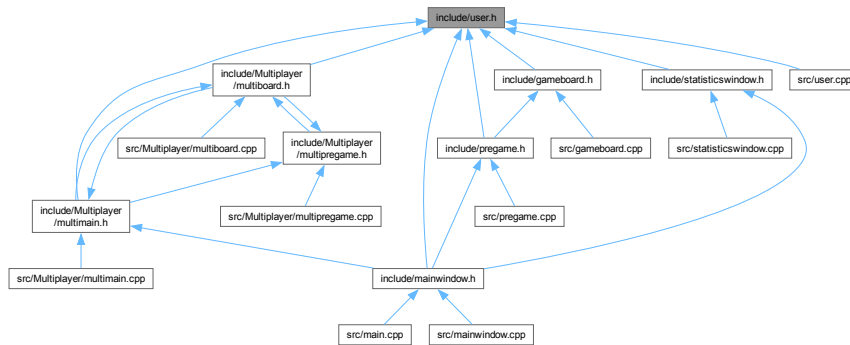
#include <QComboBox>
#include <QCryptographicHash>
#include <QDebug>
#include <QDir>
#include <QFile>
#include <QJsonDocument>
#include <QJsonObject>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QStandardPaths>
#include <QVBoxLayout>
#include <QWidget>
#include "createaccountwindow.h"

```

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.

### 5.27.1 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



## 5.28 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

```

```

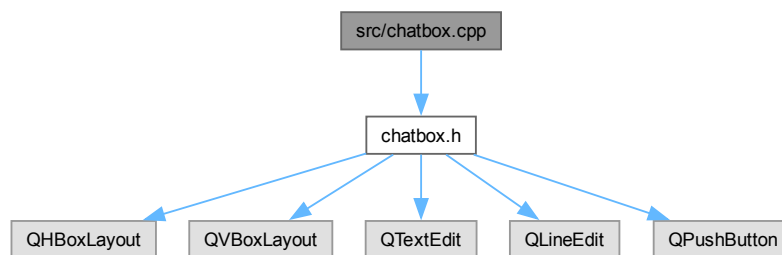
00242 private:
00249     explicit User(QWidget* parent = nullptr);
00250
00255     CreateAccountWindow* createAccountWindow;
00256
00261     QString jsonFilePath = "resources/profile.json";
00262
00267     QPushButton* backButton;
00268
00273     QPushButton* createAccountButton;
00274
00279     QComboBox*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

```

## 5.29 src/chatbox.cpp File Reference

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

Include dependency graph for chatbox.cpp:

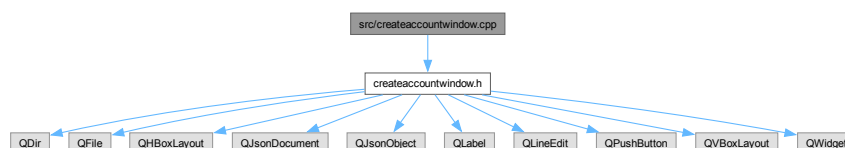


## 5.30 src/createaccountwindow.cpp File Reference

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

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

Include dependency graph for createaccountwindow.cpp:





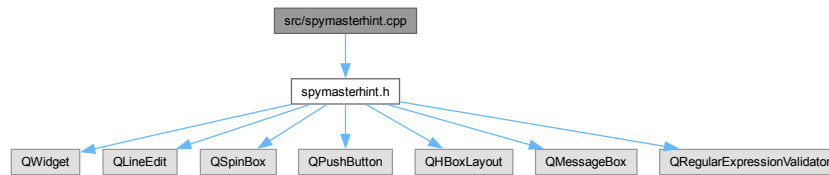




## 5.39 src/spymasterhint.cpp File Reference

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

Include dependency graph for spymasterhint.cpp:

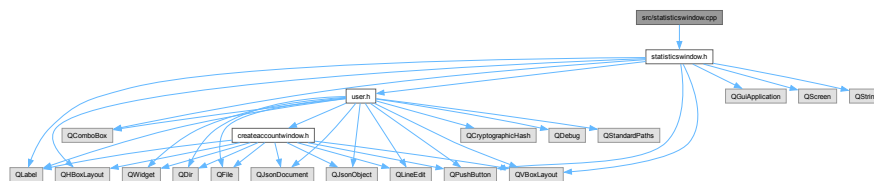


## 5.40 src/statisticswindow.cpp File Reference

The screen to show the user's statistics.

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

Include dependency graph for statisticswindow.cpp:



### 5.40.1 Detailed Description

The screen to show the user's statistics.

**Author**

Team 9 - UWO CS 3307

**Version**

0.1

**Date**

2025-03-30

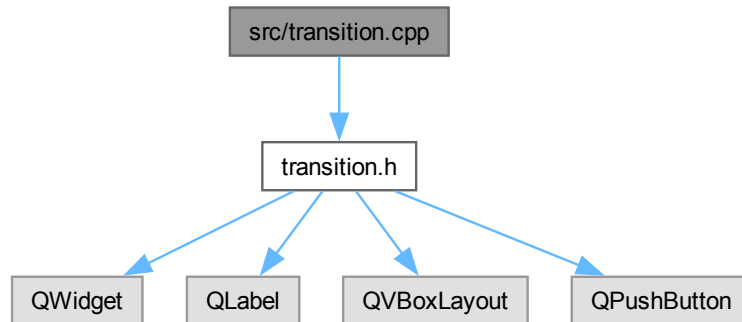
**Copyright**

Copyright (c) 2025

## 5.41 src/transition.cpp File Reference

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

Include dependency graph for transition.cpp:



## 5.42 src/tutorial.cpp File Reference

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

Include dependency graph for tutorial.cpp:

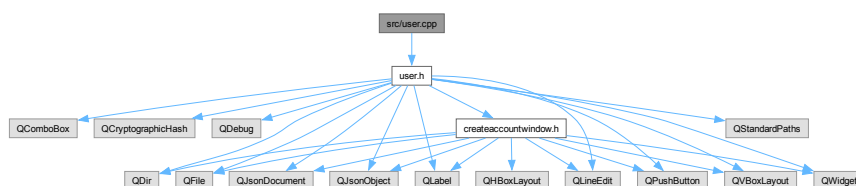


## 5.43 src/user.cpp File Reference

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

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

Include dependency graph for user.cpp:



### 5.43.1 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, [11](#)
- ~GameBoard
  - GameBoard, [23](#)
- ~MainWindow
  - MainWindow, [35](#)
- ~MultiMain
  - MultiMain, [57](#)
- ~MultiPregame
  - MultiPregame, [65](#)
- ~OperatorGuess
  - OperatorGuess, [73](#)
- ~PreGame
  - PreGame, [77](#)
- ~SpymasterHint
  - SpymasterHint, [83](#)
- ~StatisticsWindow
  - StatisticsWindow, [88](#)
- ~Transition
  - Transition, [92](#)
- ~Tutorial
  - Tutorial, [95](#)
- ~User
  - User, [100](#)
- accountCreated
  - CreateAccountWindow, [16](#)
- addPlayerMessage
  - ChatBox, [11](#)
- addSystemMessage
  - ChatBox, [12](#)
- advanceTurn
  - MultiBoard, [43](#)
- advanceTurnSpymaster
  - MultiBoard, [43](#)
- ASSASSIN
  - GameBoard, [22](#)
  - MultiBoard, [42](#)
- back
  - CreateAccountWindow, [16](#)
- backButton
  - MultiMain, [61](#)
  - PreGame, [79](#)
  - User, [106](#)
- backToMainButton
  - StatisticsWindow, [89](#)
- backToMainMenu
  - User, [100](#)
- backToMainWindow
  - MultiMain, [57](#)
  - PreGame, [77](#)
  - StatisticsWindow, [88](#)
- backToMultiMain
  - MultiPregame, [66](#)
- BLUE\_OP
  - GameBoard, [23](#)
  - MultiBoard, [43](#)
- BLUE\_SPY
  - GameBoard, [23](#)
  - MultiBoard, [43](#)
- BLUE\_TEAM
  - ChatBox, [11](#)
  - GameBoard, [22](#)
  - MultiBoard, [42](#)
- blueCardsRemaining
  - GameBoard, [29](#)
  - MultiBoard, [50](#)
- blueCardText
  - MultiBoard, [50](#)
- blueOperativeName
  - GameBoard, [29](#)
- blueScoreLabel
  - GameBoard, [29](#)
- blueSpyMasterName
  - GameBoard, [29](#)
- blueTeamLabel
  - GameBoard, [29](#)
- blueTeamLayout
  - PreGame, [79](#)
- blueTeamOperativeComboBox
  - PreGame, [79](#)
- blueTeamSpyMasterComboBox
  - PreGame, [80](#)
- buttonsLayout
  - PreGame, [80](#)
- cards
  - GameBoard, [30](#)
  - MultiBoard, [50](#)
- CardType
  - GameBoard, [22](#)
  - MultiBoard, [42](#)
- centralWidget
  - MainWindow, [36](#)
  - Tutorial, [96](#)
- ChatBox, [9](#)
  - ~ChatBox, [11](#)
  - addPlayerMessage, [11](#)
  - addSystemMessage, [12](#)

- BLUE\_TEAM, 11
- ChatBox, 11
- chatDisplay, 13
- chatInput, 13
- clearChat, 12
- limitReachedMessage, 12
- massSend, 12
- playerName, 13
- RED\_TEAM, 11
- sendButton, 13
- sendMessage, 13
- setPlayerName, 13
- Team, 10
- team, 14
- chatBox
  - GameBoard, 30
  - MultiBoard, 50
- chatDisplay
  - ChatBox, 13
- chatInput
  - ChatBox, 13
- checkGameEnd
  - GameBoard, 24
  - MultiBoard, 44
- clearChat
  - ChatBox, 12
- clearUI
  - MultiPregame, 66
- clickCount
  - Tutorial, 96
- closeEvent
  - Tutorial, 95
- continueButton
  - Transition, 93
  - Tutorial, 96
- continueClicked
  - Transition, 92
- correspondingNumber
  - GameBoard, 30
  - MultiBoard, 50
- createAccountButton
  - CreateAccountWindow, 18
  - MainWindow, 36
  - PreGame, 80
  - User, 106
- CreateAccountWindow, 14
  - accountCreated, 16
  - back, 16
  - createAccountButton, 18
  - CreateAccountWindow, 16
  - getInstance, 16
  - goBack, 17
  - instance, 18
  - jsonFilePath, 18
  - onCreateAccountClicked, 17
  - previousScreen, 18
  - saveJsonFile, 17
  - setPreviousScreen, 17
  - show, 17
  - statusLabel, 18
  - usernameEdit, 18
- createAccountWindow
  - MainWindow, 36
  - PreGame, 80
  - User, 106
- createRoomButton
  - MultiMain, 61
- currentGuesses
  - GameBoard, 30
- currentHint
  - GameBoard, 30
  - MultiBoard, 51
- currentPlayerName
  - GameBoard, 30
- currentPlayerTeam
  - GameBoard, 30
- currentTurn
  - GameBoard, 30
- currentTurnLabel
  - GameBoard, 31
- displayGuess
  - GameBoard, 24
- displayHint
  - GameBoard, 24
  - MultiBoard, 44
- endGame
  - GameBoard, 24
  - MultiBoard, 44
- enterPregameAsClient
  - MultiMain, 57
  - MultiPregame, 66
- enterPregameAsHost
  - MultiMain, 58
  - MultiPregame, 66
- GameBoard, 19
  - ~GameBoard, 23
  - ASSASSIN, 22
  - BLUE\_OP, 23
  - BLUE\_SPY, 23
  - BLUE\_TEAM, 22
  - blueCardsRemaining, 29
  - blueOperativeName, 29
  - blueScoreLabel, 29
  - blueSpyMasterName, 29
  - blueTeamLabel, 29
  - cards, 30
  - CardType, 22
  - chatBox, 30
  - checkGameEnd, 24
  - correspondingNumber, 30
  - currentGuesses, 30
  - currentHint, 30
  - currentPlayerName, 30
  - currentPlayerTeam, 30

- currentTurn, 30
- currentTurnLabel, 31
- displayGuess, 24
- displayHint, 24
- endGame, 24
- GameBoard, 23
- gameEnded, 25
- gameGrid, 31
- generateGameGrid, 25
- GRID\_SIZE, 31
- gridLayout, 31
- loadWordsFromFile, 25
- maxGuesses, 31
- NEUTRAL, 22
- nextTurn, 25
- onCardClicked, 26
- onContinueClicked, 26
- operatorGuess, 31
- RED\_OP, 23
- RED\_SPY, 23
- RED\_TEAM, 22
- redCardsRemaining, 31
- redOperativeName, 31
- redScoreLabel, 32
- redSpyMasterName, 32
- redTeamLabel, 32
- resetGame, 26
- setBlueOperativeName, 26
- setBlueSpyMasterName, 27
- setRedOperativeName, 27
- setRedSpyMasterName, 27
- setupUI, 28
- show, 28
- showTransition, 28
- sPYmasterHint, 32
- transition, 32
- Turn, 22
- updateScores, 28
- updateTeamLabels, 29
- users, 32
- wordList, 32
- gameBoard
  - PreGame, 80
- GameBoard::Card, 7
  - revealed, 7
  - type, 7
  - word, 7
- gameEnded
  - GameBoard, 25
- gameGrid
  - GameBoard, 31
  - MultiBoard, 51
- gamesPlayedStats
  - StatisticsWindow, 89
- gameStarted
  - MultiPregame, 67
- gamesWinRateStats
  - StatisticsWindow, 89
- gamesWinStats
  - StatisticsWindow, 89
- gameVerticalLayout
  - MultiBoard, 51
- generateGameGrid
  - GameBoard, 25
  - MultiBoard, 45
- getBlueTeamOperativeNickname
  - PreGame, 77
- getBlueTeamSpyMasterNickname
  - PreGame, 77
- getColorStyle
  - MultiBoard, 45
- getGamesPlayed
  - User, 100
- getGuessHit
  - User, 101
- getGuessTotal
  - User, 101
- getHitRate
  - User, 101
- getInstance
  - CreateAccountWindow, 16
- getMyTeam
  - MultiBoard, 45
- getRedTeamOperativeNickname
  - PreGame, 78
- getRedTeamSpyMasterNickname
  - PreGame, 78
- getWinRate
  - User, 102
- getWins
  - User, 102
- giveClueButton
  - SpyMasterHint, 85
- goBack
  - CreateAccountWindow, 17
  - MultiBoard, 46
- goBackToMain
  - PreGame, 78
  - StatisticsWindow, 88
- GRID\_SIZE
  - GameBoard, 31
  - MultiBoard, 51
- gridLayout
  - GameBoard, 31
- guess
  - MultiBoard, 51
- guessHitRateStats
  - StatisticsWindow, 89
- guessHitStats
  - StatisticsWindow, 89
- guessSubmitted
  - OperatorGuess, 73
- guessTotalStats
  - StatisticsWindow, 90
- handleCreateAccount
  - User, 102

- handleGameEnd
  - PreGame, 78
- handleLogin
  - User, 102
- handleNewConnection
  - MultiBoard, 46
- handleRoleSelection
  - MultiPregame, 67
- handleTileClick
  - MultiBoard, 46
- hint
  - MultiBoard, 51
- hintLineEdit
  - SpymasterHint, 85
- hintSubmitted
  - SpymasterHint, 84
- hit
  - User, 102
- include/chatbox.h, 107, 109
- include/createaccountwindow.h, 109, 111
- include/gameboard.h, 111, 113
- include/mainwindow.h, 114, 115
- include/Multiplayer/multiboard.h, 116, 118
- include/Multiplayer/multimain.h, 120, 121
- include/Multiplayer/multipregame.h, 122, 123
- include/operatorguess.h, 124, 125
- include/pregame.h, 126, 128
- include/spymasterhint.h, 129, 130
- include/statisticswindow.h, 131, 132
- include/transition.h, 133, 135
- include/tutorial.h, 135, 136
- include/user.h, 137, 139
- initializeBoardColors
  - MultiBoard, 46
- initializeWords
  - MultiBoard, 47
- instance
  - CreateAccountWindow, 18
  - User, 103
- isMyTurn
  - MultiBoard, 47
- joinRoomButton
  - MultiMain, 61
- jsonContentLabel
  - User, 106
- jsonFilePath
  - CreateAccountWindow, 18
  - User, 106
- label
  - PreGame, 80
- layout
  - MainWindow, 36
  - PreGame, 80
- limitReachedMessage
  - ChatBox, 12
- loadJsonFile
  - User, 103
- loadWordsFromFile
  - GameBoard, 25
  - MultiBoard, 47
- localPlayButton
  - MainWindow, 37
- loginButton
  - User, 106
- lost
  - User, 103
- m\_checked
  - MultiPregame, 70
- m\_clients
  - MultiBoard, 51
  - MultiMain, 61
  - MultiPregame, 70
- m\_clientSocket
  - MultiBoard, 51
  - MultiMain, 61
  - MultiPregame, 70
- m\_currentRole
  - MultiBoard, 52
- m\_currentTurnIndex
  - MultiBoard, 52
- m\_currentUsername
  - MultiBoard, 52
- m\_grid
  - MultiBoard, 52
- m\_isHost
  - MultiBoard, 52
  - MultiPregame, 70
- m\_playerInfoLabel
  - MultiBoard, 52
- m\_playerRoles
  - MultiBoard, 52
- m\_pregame
  - MultiBoard, 52
- m\_roles
  - MultiPregame, 70
- m\_server
  - MultiBoard, 53
  - MultiMain, 61
  - MultiPregame, 70
- m\_tileColors
  - MultiBoard, 53
- m\_tiles
  - MultiBoard, 53
- m\_turnLabel
  - MultiBoard, 53
- m\_turnOrder
  - MultiBoard, 53
- m\_username
  - MultiMain, 62
  - MultiPregame, 70
- m\_usernames
  - MultiMain, 62
  - MultiPregame, 70
- m\_words

- MultiBoard, 53
- main
  - main.cpp, 142
  - MultiBoard, 53
- main.cpp
  - main, 142
- mainLayout
  - MultiBoard, 53
- MainWindow, 33
  - ~MainWindow, 35
  - centralWidget, 36
  - createAccountButton, 36
  - createAccountWindow, 36
  - layout, 36
  - localPlayButton, 37
  - MainWindow, 35
  - multiMain, 37
  - onlineGameWindow, 37
  - onlinePlayButton, 37
  - openCreateAccount, 35
  - openMultiMain, 35
  - openOnlineGame, 35
  - openPreGame, 35
  - openStatsWindow, 36
  - openTutorial, 36
  - preGameWindow, 37
  - showMainWindow, 36
  - statsButton, 37
  - statsWindow, 37
  - titleLabel, 37
  - tutorialButton, 38
  - tutorialWindow, 38
- massSend
  - ChatBox, 12
- maxGuesses
  - GameBoard, 31
- messageLabel
  - Transition, 93
- miss
  - User, 103
- MultiBoard, 38
  - advanceTurn, 43
  - advanceTurnSpymaster, 43
  - ASSASSIN, 42
  - BLUE\_OP, 43
  - BLUE\_SPY, 43
  - BLUE\_TEAM, 42
  - blueCardsRemaining, 50
  - blueCardText, 50
  - cards, 50
  - CardType, 42
  - chatBox, 50
  - checkGameEnd, 44
  - correspondingNumber, 50
  - currentHint, 51
  - displayHint, 44
  - endGame, 44
  - gameGrid, 51
  - gameVerticalLayout, 51
  - generateGameGrid, 45
  - getColorStyle, 45
  - getMyTeam, 45
  - goBack, 46
  - GRID\_SIZE, 51
  - guess, 51
  - handleNewConnection, 46
  - handleTileClick, 46
  - hint, 51
  - initializeBoardColors, 46
  - initializeWords, 47
  - isMyTurn, 47
  - loadWordsFromFile, 47
  - m\_clients, 51
  - m\_clientSocket, 51
  - m\_currentRole, 52
  - m\_currentTurnIndex, 52
  - m\_currentUsername, 52
  - m\_grid, 52
  - m\_isHost, 52
  - m\_playerInfoLabel, 52
  - m\_playerRoles, 52
  - m\_pregame, 52
  - m\_server, 53
  - m\_tileColors, 53
  - m\_tiles, 53
  - m\_turnLabel, 53
  - m\_turnOrder, 53
  - m\_words, 53
  - main, 53
  - mainLayout, 53
  - MultiBoard, 43
  - NEUTRAL, 42
  - processChatMessage, 47
  - processMessage, 48
  - RED\_OP, 43
  - RED\_SPY, 43
  - RED\_TEAM, 42
  - redCardsRemaining, 54
  - redCardText, 54
  - revealTile, 48
  - sendInitialGameState, 48
  - sendToAll, 49
  - setupBoard, 49
  - setupUI, 49
  - socketDisconnected, 49
  - Turn, 42
  - updateTurnDisplay, 50
  - users, 54
  - wordList, 54
- MultiBoard::Card, 8
  - revealed, 8
  - type, 8
  - word, 8
- MultiMain, 55
  - ~MultiMain, 57
  - backButton, 61

- backToMainWindow, 57
- createRoomButton, 61
- enterPregameAsClient, 57
- enterPregameAsHost, 58
- joinRoomButton, 61
- m\_clients, 61
- m\_clientSocket, 61
- m\_server, 61
- m\_username, 62
- m\_usernames, 62
- MultiMain, 57
- onConnected, 58
- onCreateRoomClicked, 58
- onDisconnected, 58
- onJoinRoomClicked, 59
- onNewConnection, 59
- openMainWindow, 59
- processTextMessage, 59
- sendLobbyListToAll, 60
- showMainWindow, 60
- socketDisconnected, 60
- titleLabel, 62
- updateLobbyList, 60
- multiMain
  - MainWindow, 37
- MultiPregame, 62
  - ~MultiPregame, 65
  - backToMultiMain, 66
  - clearUI, 66
  - enterPregameAsClient, 66
  - enterPregameAsHost, 66
  - gameStarted, 67
  - handleRoleSelection, 67
  - m\_checked, 70
  - m\_clients, 70
  - m\_clientSocket, 70
  - m\_isHost, 70
  - m\_roles, 70
  - m\_server, 70
  - m\_username, 70
  - m\_usernames, 70
  - MultiPregame, 65
  - onNewConnection, 67
  - playerList, 71
  - processMessage, 68
  - resetUIState, 68
  - sendLobbyUpdate, 68
  - setupUI, 68
  - showPregame, 69
  - socketDisconnected, 69
  - startGame, 69
- NEUTRAL
  - GameBoard, 22
  - MultiBoard, 42
- nextTurn
  - GameBoard, 25
- numberSpinBox
  - SpymasterHint, 85
- onCardClicked
  - GameBoard, 26
- onConnected
  - MultiMain, 58
- onContinueClicked
  - GameBoard, 26
  - Tutorial, 96
- onCreateAccountClicked
  - CreateAccountWindow, 17
- onCreateRoomClicked
  - MultiMain, 58
- onDisconnected
  - MultiMain, 58
- onJoinRoomClicked
  - MultiMain, 59
- onlineGameWindow
  - MainWindow, 37
- onlinePlayButton
  - MainWindow, 37
- onNewConnection
  - MultiMain, 59
  - MultiPregame, 67
- openCreateAccount
  - MainWindow, 35
  - PreGame, 78
- openMainWindow
  - MultiMain, 59
- openMultiMain
  - MainWindow, 35
- openOnlineGame
  - MainWindow, 35
- openPreGame
  - MainWindow, 35
- openStatsWindow
  - MainWindow, 36
- openTutorial
  - MainWindow, 36
- OperatorGuess, 71
  - ~OperatorGuess, 73
  - guessSubmitted, 73
  - OperatorGuess, 72
  - reset, 73
  - submitGuess, 73
  - submitGuessButton, 74
- operatorGuess
  - GameBoard, 31
- playerList
  - MultiPregame, 71
- playerName
  - ChatBox, 13
- populateDropDown
  - StatisticsWindow, 88
- populateUserDropdowns
  - PreGame, 78
- populateUsernameComboBox
  - User, 104
- PreGame, 74
  - ~PreGame, 77

- backButton, 79
- backToMainWindow, 77
- blueTeamLayout, 79
- blueTeamOperativeComboBox, 79
- blueTeamSpyMasterComboBox, 80
- buttonsLayout, 80
- createAccountButton, 80
- createAccountWindow, 80
- gameBoard, 80
- getBlueTeamOperativeNickname, 77
- getBlueTeamSpyMasterNickname, 77
- getRedTeamOperativeNickname, 78
- getRedTeamSpyMasterNickname, 78
- goBackToMain, 78
- handleGameEnd, 78
- label, 80
- layout, 80
- openCreateAccount, 78
- populateUserDropdowns, 78
- PreGame, 77
- redTeamLayout, 80
- redTeamOperativeComboBox, 81
- redTeamSpyMasterComboBox, 81
- show, 79
- start, 79
- startButton, 81
- startGame, 79
- teamsLayout, 81
- update, 79
- usernames, 81
- users, 81
- preGameWindow
  - MainWindow, 37
- previousScreen
  - CreateAccountWindow, 18
- processChatMessage
  - MultiBoard, 47
- processMessage
  - MultiBoard, 48
  - MultiPregame, 68
- processTextMessage
  - MultiMain, 59
- RED\_OP
  - GameBoard, 23
  - MultiBoard, 43
- RED\_SPY
  - GameBoard, 23
  - MultiBoard, 43
- RED\_TEAM
  - ChatBox, 11
  - GameBoard, 22
  - MultiBoard, 42
- redCardsRemaining
  - GameBoard, 31
  - MultiBoard, 54
- redCardText
  - MultiBoard, 54
- redOperativeName
  - GameBoard, 31
- redScoreLabel
  - GameBoard, 32
- redSpyMasterName
  - GameBoard, 32
- redTeamLabel
  - GameBoard, 32
- redTeamLayout
  - PreGame, 80
- redTeamOperativeComboBox
  - PreGame, 81
- redTeamSpyMasterComboBox
  - PreGame, 81
- refreshUserDropdown
  - User, 104
- renameUser
  - User, 104
- reset
  - OperatorGuess, 73
  - SpymasterHint, 84
- resetGame
  - GameBoard, 26
- resetTutorial
  - Tutorial, 96
- resetUIState
  - MultiPregame, 68
- revealed
  - GameBoard::Card, 7
  - MultiBoard::Card, 8
- revealTile
  - MultiBoard, 48
- saveJsonFile
  - CreateAccountWindow, 17
- sendButton
  - ChatBox, 13
- sendInitialGameState
  - MultiBoard, 48
- sendLobbyListToAll
  - MultiMain, 60
- sendLobbyUpdate
  - MultiPregame, 68
- sendMessage
  - ChatBox, 13
- sendToAll
  - MultiBoard, 49
- setBlueOperativeName
  - GameBoard, 26
- setBlueSpyMasterName
  - GameBoard, 27
- setMessage
  - Transition, 92
- setPlayerName
  - ChatBox, 13
- setPreviousScreen
  - CreateAccountWindow, 17
- setRedOperativeName
  - GameBoard, 27
- setRedSpyMasterName

- GameBoard, 27
- setupBoard
  - MultiBoard, 49
- setupUI
  - GameBoard, 28
  - MultiBoard, 49
  - MultiPregame, 68
- show
  - CreateAccountWindow, 17
  - GameBoard, 28
  - PreGame, 79
  - StatisticsWindow, 88
  - User, 104
- showMainMenu
  - User, 104
- showMainWindow
  - MainWindow, 36
  - MultiMain, 60
- showPregame
  - MultiPregame, 69
- showTransition
  - GameBoard, 28
- showUserStats
  - StatisticsWindow, 89
- showUserStatsButton
  - StatisticsWindow, 90
- socketDisconnected
  - MultiBoard, 49
  - MultiMain, 60
  - MultiPregame, 69
- SpymasterHint, 82
  - ~SpymasterHint, 83
  - giveClueButton, 85
  - hintLineEdit, 85
  - hintSubmitted, 84
  - numberSpinBox, 85
  - reset, 84
  - SpymasterHint, 83
  - submitHint, 84
  - textToUppercase, 84
  - textValidator, 85
  - updateButtonClickable, 85
- spymasterHint
  - GameBoard, 32
- src/chatbox.cpp, 140
- src/createaccountwindow.cpp, 140
- src/gameboard.cpp, 141
- src/main.cpp, 141
- src/mainwindow.cpp, 142
- src/Multiplayer/multiboard.cpp, 142
- src/Multiplayer/multimain.cpp, 142
- src/Multiplayer/multipregame.cpp, 142
- src/operatorguess.cpp, 143
- src/pregame.cpp, 143
- src/spymasterhint.cpp, 144
- src/statisticswindow.cpp, 144
- src/transition.cpp, 145
- src/tutorial.cpp, 145
- src/user.cpp, 145
- start
  - PreGame, 79
- startButton
  - PreGame, 81
- startGame
  - MultiPregame, 69
  - PreGame, 79
- StatisticsWindow, 86
  - ~StatisticsWindow, 88
  - backToMainButton, 89
  - backToMainWindow, 88
  - gamesPlayedStats, 89
  - gamesWinRateStats, 89
  - gamesWinStats, 89
  - goBackToMain, 88
  - guessHitRateStats, 89
  - guessHitStats, 89
  - guessTotalStats, 90
  - populateDropDown, 88
  - show, 88
  - showUserStats, 89
  - showUserStatsButton, 90
  - StatisticsWindow, 88
  - username, 90
  - usernameComboBox, 90
  - usernameTitle, 90
  - users, 90
- statsButton
  - MainWindow, 37
- statsWindow
  - MainWindow, 37
- statusLabel
  - CreateAccountWindow, 18
- submitGuess
  - OperatorGuess, 73
- submitGuessButton
  - OperatorGuess, 74
- submitHint
  - SpymasterHint, 84
- Team
  - ChatBox, 10
- team
  - ChatBox, 14
- teamsLayout
  - PreGame, 81
- textBox
  - Tutorial, 97
- textToUppercase
  - SpymasterHint, 84
- textValidator
  - SpymasterHint, 85
- titleLabel
  - MainWindow, 37
  - MultiMain, 62
  - Tutorial, 97
- Transition, 91
  - ~Transition, 92



- continueButton, 93
- continueClicked, 92
- messageLabel, 93
- setMessage, 92
- Transition, 92
- transition
  - GameBoard, 32
- Turn
  - GameBoard, 22
  - MultiBoard, 42
- Tutorial, 93
  - ~Tutorial, 95
  - centralWidget, 96
  - clickCount, 96
  - closeEvent, 95
  - continueButton, 96
  - onContinueClicked, 96
  - resetTutorial, 96
  - textBox, 97
  - titleLabel, 97
  - Tutorial, 95
  - tutorialClosed, 96
  - updateContinueButtonPosition, 96
- tutorialButton
  - MainWindow, 38
- tutorialClosed
  - Tutorial, 96
- tutorialWindow
  - MainWindow, 38
- type
  - GameBoard::Card, 7
  - MultiBoard::Card, 8
- update
  - PreGame, 79
- updateButtonClickable
  - SpymasterHint, 85
- updateContinueButtonPosition
  - Tutorial, 96
- updateGamesPlayed
  - User, 104
- updateGuessHit
  - User, 105
- updateGuessTotal
  - User, 105
- updateLobbyList
  - MultiMain, 60
- updateScores
  - GameBoard, 28
- updateTeamLabels
  - GameBoard, 29
- updateTurnDisplay
  - MultiBoard, 50
- updateWins
  - User, 105
- User, 97
  - ~User, 100
  - backButton, 106
  - backToMainMenu, 100
  - createAccountButton, 106
  - createAccountWindow, 106
  - getGamesPlayed, 100
  - getGuessHit, 101
  - getGuessTotal, 101
  - getHitRate, 101
  - getWinRate, 102
  - getWins, 102
  - handleCreateAccount, 102
  - handleLogin, 102
  - hit, 102
  - instance, 103
  - jsonContentLabel, 106
  - jsonFilePath, 106
  - loadJsonFile, 103
  - loginButton, 106
  - lost, 103
  - miss, 103
  - populateUsernameComboBox, 104
  - refreshUserDropdown, 104
  - renameUser, 104
  - show, 104
  - showMainMenu, 104
  - updateGamesPlayed, 104
  - updateGuessHit, 105
  - updateGuessTotal, 105
  - updateWins, 105
  - User, 100
  - usernameComboBox, 106
  - won, 105
- username
  - StatisticsWindow, 90
- usernameComboBox
  - StatisticsWindow, 90
  - User, 106
- usernameEdit
  - CreateAccountWindow, 18
- usernames
  - PreGame, 81
- usernameTitle
  - StatisticsWindow, 90
- users
  - GameBoard, 32
  - MultiBoard, 54
  - PreGame, 81
  - StatisticsWindow, 90
- won
  - User, 105
- word
  - GameBoard::Card, 7
  - MultiBoard::Card, 8
- wordList
  - GameBoard, 32
  - MultiBoard, 54