C++Names

Generated by Doxygen 1.13.2

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

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:

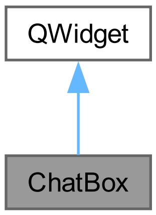
- include/Multiplayer/multiboard.h

4.3 ChatBox Class Reference

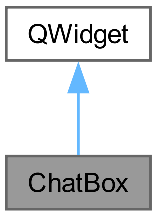
A widget for the chat feature in the game.

```
#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 QPush↩ Button 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

playerName	The name of the player using this chat box.
team	The team of the player (red or blue).
parent	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

playerName	The name of the player sending the message.
message	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

message	The system message text.
team	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

playerName	The name of the player sending the message.
message	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

name	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:

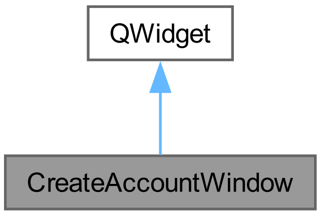
- include/chatbox.h
- src/chatbox.cpp

4.4 CreateAccountWindow Class Reference

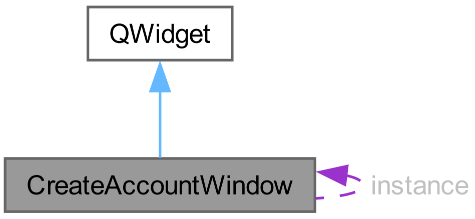
The CreateAccountWindow class provides a singleton interface for creating new user accounts This window allows users to input a username and creates a profile JSON file for the new account.

```
#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

parent	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

parent	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

username	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

previous	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:

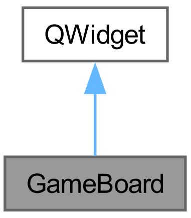
- include/createaccountwindow.h
- src/createaccountwindow.cpp

4.5 GameBoard Class Reference

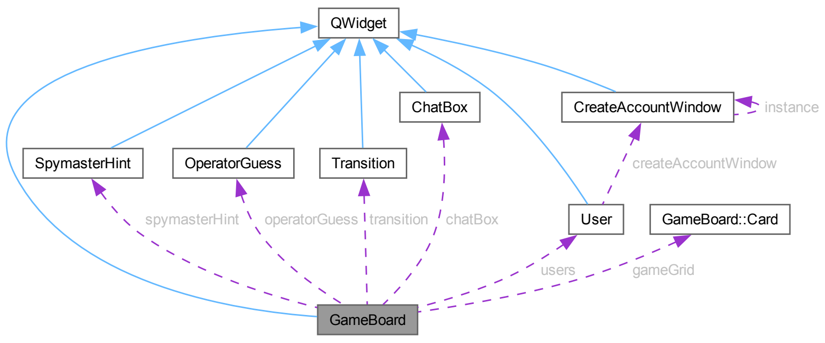
A class representing the game board for the Spy Master game.

```
#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

redSpyMaster	The name of the red team's spymaster.
redOperative	The name of the red team's operative.
blueSpyMaster	The name of the blue team's spymaster.
blueOperative	The name of the blue team's operative.
parent	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

hint	The hint to be displayed.
number	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

message	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

row	The row of the clicked card.
col	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

name	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

name	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

name	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

name	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:

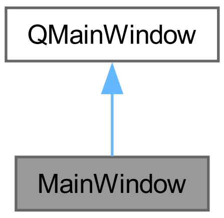
- include/gameboard.h
- src/gameboard.cpp

4.6 MainWindow Class Reference

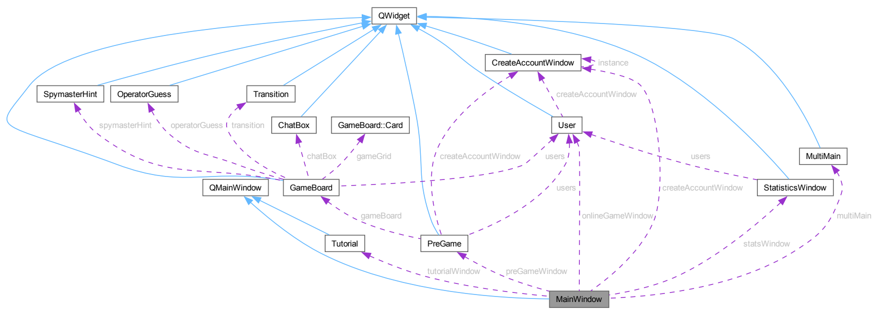
The main application window.

```
#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

parent	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:

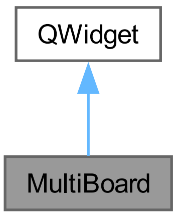
- include/mainwindow.h
- src/mainwindow.cpp

4.7 MultiBoard Class Reference

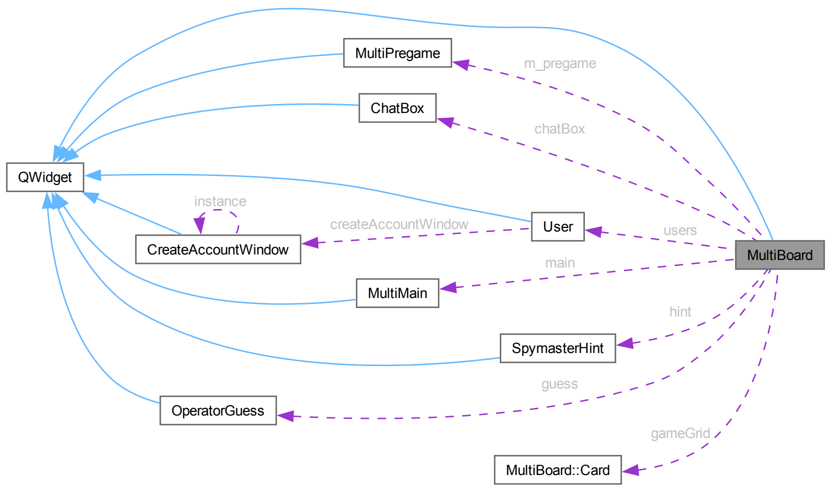
A widget that implements the multiplayer game board for a team-based word guessing game.

```
#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	Card belonging to the red team
BLUE_TEAM	Card 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

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

hint	The word hint provided by the spymaster.
number	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

hint	The word hint to display.
number	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

message	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

color	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

playerName	The name of the player who sent the message.
message	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

message	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

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

Author

Group 9

4.7.4.19 sendInitialGameState()

```
void MultiBoard::sendInitialGameState ()  [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

message	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:

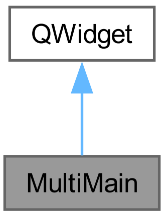
- include/Multiplayer/multiboard.h
- src/Multiplayer/multiboard.cpp

4.8 MultiMain Class Reference

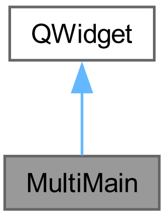
A widget that implements the main multiplayer lobby for creating and joining game rooms.

```
#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

parent	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

socket	Pointer to the client's WebSocket connection.
username	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

server	Pointer to the WebSocket server instance.
username	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

message	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:

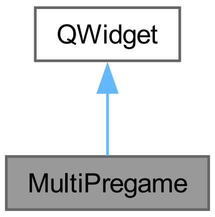
- include/Multiplayer/multimain.h
- src/Multiplayer/multimain.cpp

4.9 MultiPregame Class Reference

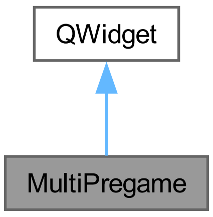
A widget that implements the pre-game lobby for multiplayer games.

```
#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

server	Pointer to the WebSocket server for hosting the game.
username	The username of the host player.
parent	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

socket	Pointer to the WebSocket connection to the host.
username	The username of the client player.
parent	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

socket	Pointer to the client's WebSocket connection.
username	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

server	Pointer to the WebSocket server instance.
username	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

isHost	Boolean indicating if this instance is the host.
server	Pointer to the WebSocket server (if host).
clients	List of connected client WebSockets.
clientSocket	Pointer to this player's WebSocket (if client).
playerRoles	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

message	The role selection message.
sender	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

message	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:

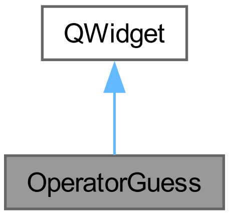
- include/Multiplayer/multipregame.h
- src/Multiplayer/multipregame.cpp

4.10 OperatorGuess Class Reference

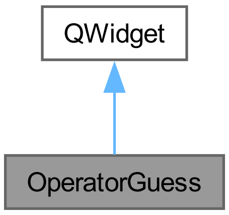
A widget that provides the interface for operators to submit guesses during gameplay.

`#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

parent	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:

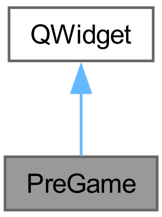
- include/operatorguess.h
- src/operatorguess.cpp

4.11 PreGame Class Reference

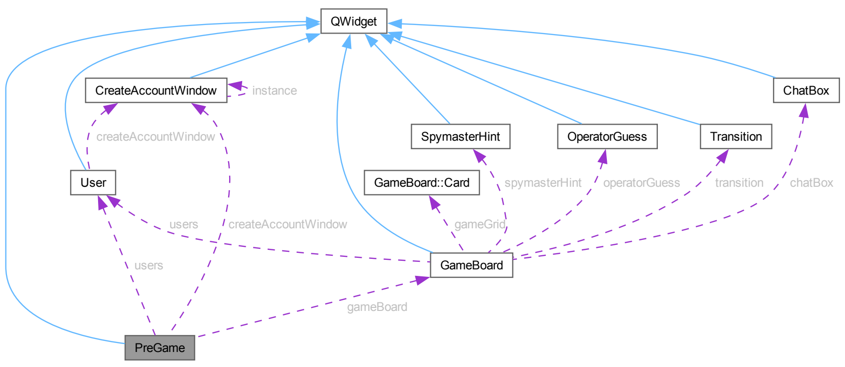
The PreGame class provides the interface for setting up a new game This includes selecting players for each team and role before starting the game.

```
#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

parent	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:

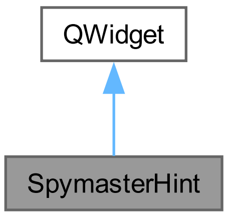
- include/pregame.h
- src/pregame.cpp

4.12 SpymasterHint Class Reference

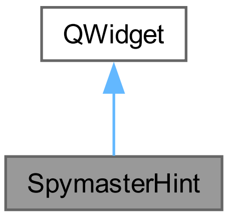
A widget for the spymaster to input a hint and the number of words associated with it.

```
#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

parent	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

hint	The hint text.
number	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

text	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:

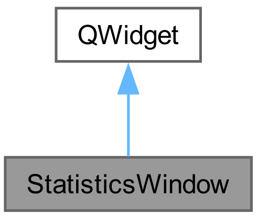
- include/spymasterhint.h
- src/spymasterhint.cpp

4.13 StatisticsWindow Class Reference

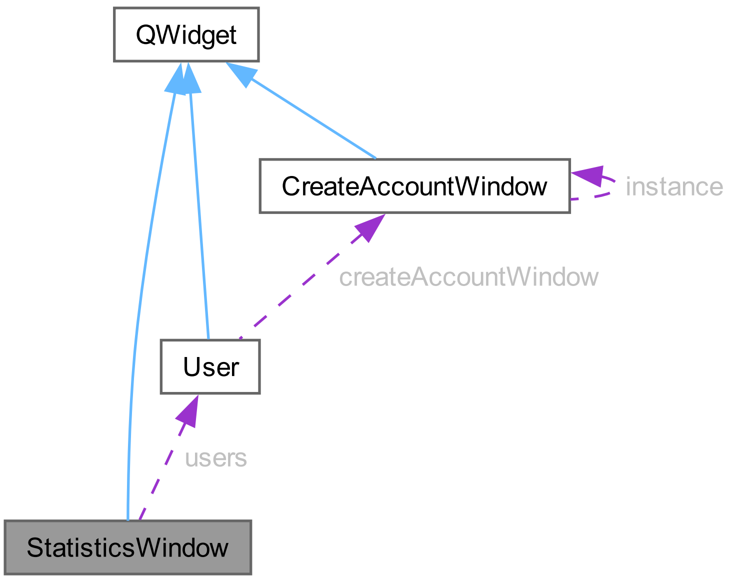
The class that shows the Statistics screen Displays game statistics for selected users including win rates and guess accuracy.

`#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

parent	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:

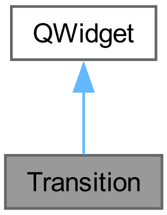
- include/statisticswindow.h
- src/statisticswindow.cpp

4.14 Transition Class Reference

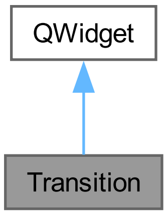
A widget for displaying a transition message and a button to continue.

```
#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

parent	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

message	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:

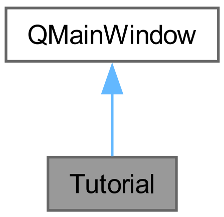
- include/transition.h
- src/transition.cpp

4.15 Tutorial Class Reference

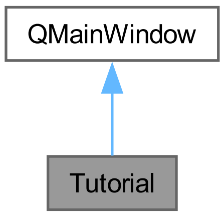
The tutorial window that guides users through the game mechanics.

```
#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

parent	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

event	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:

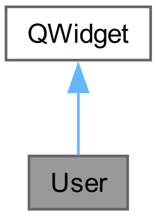
- include/tutorial.h
- src/tutorial.cpp

4.16 User Class Reference

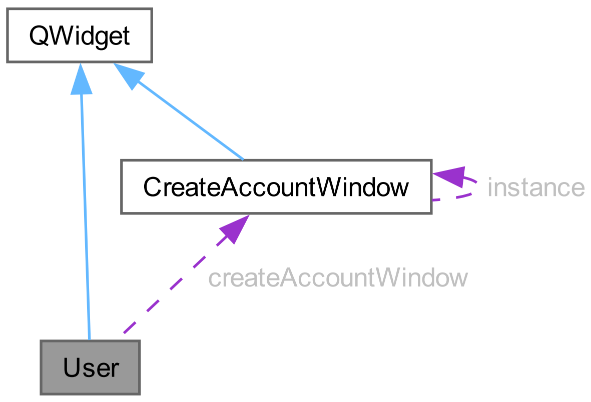
User class to handle local log in and loading/storing json files. This is a singleton class to ensure only one instance of user management exists. Manages user profiles, statistics, and authentication.

```
#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

parent	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

username	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

username	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

username	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

username	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

username	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

username	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

username	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

parent	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

username	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

username	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

jsonObject	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

oldUsername	old username of the user
newUsername	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

username	username of the user to update
newGamesPlayed	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

username	username of the user
newGuessHit	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

username	username of the user
newGuessTotal	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

username	username of the user
newWins	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

username	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

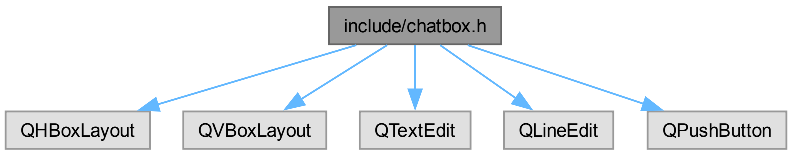
File Documentation

5.1 include/chatbox.h File Reference

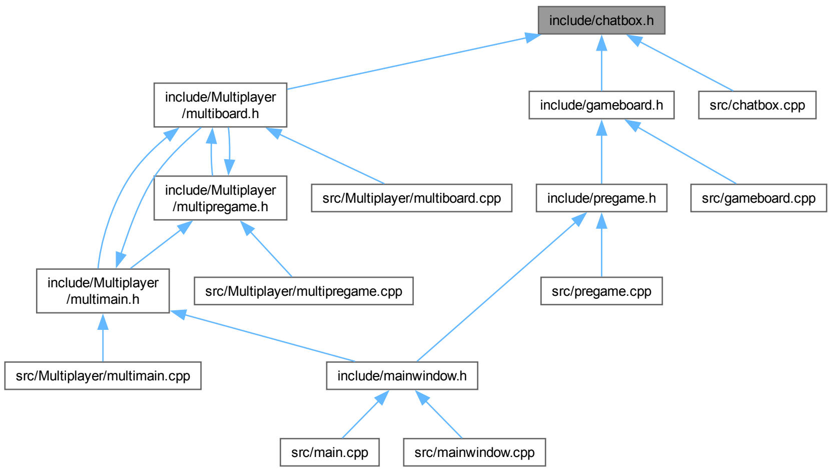
Header file for the ChatBox class, which provides a UI for the chat feature in the game.

```
#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:


include/chatbox.h
include/Multiplayer /multiboard.h
include/gameboard.h
src/chatbox.cpp
include/Multiplayer /multipregame.h
src/Multiplayer/multiboard.cpp
include/pregame.h
src/gameboard.cpp
include/Multiplayer /multimain.h
src/Multiplayer/multipregame.cpp
src/pregame.cpp
src/Multiplayer/multimain.cpp
include/mainwindow.h
src/main.cpp
src/mainwindow.cpp

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.

```

#ifndef CHATBOX_H
#define CHATBOX_H

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

class ChatBox : public QWidget {
    Q_OBJECT

public:
    enum Team {
        RED_TEAM,
        BLUE_TEAM
    };

    explicit ChatBox(const QString& playerName, Team team, QWidget* parent = nullptr);

    ~ChatBox();

    void addSystemMessage(const QString& message, Team team);

    void addPlayerMessage(const QString& playerName, const QString& message);

    void setPlayerName(const QString& name);

    void clearChat();

    void limitReachedMessage();

public slots:
    void sendMessage();
signals:
    void massSend(const QString& playerName, const QString& message);

private:
    Team team;

    QTextEdit* chatDisplay;

    QLineEdit* chatInput;

    QPushButton* sendButton;

    QString playerName;
};

#endif // CHATBOX_H
```

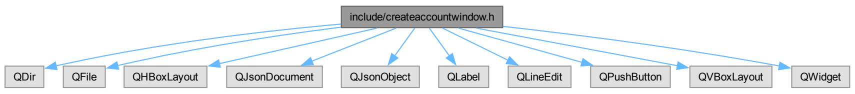
5.3 include/createaccountwindow.h File Reference

Header file for the CreateAccountWindow class which handles user account creation.

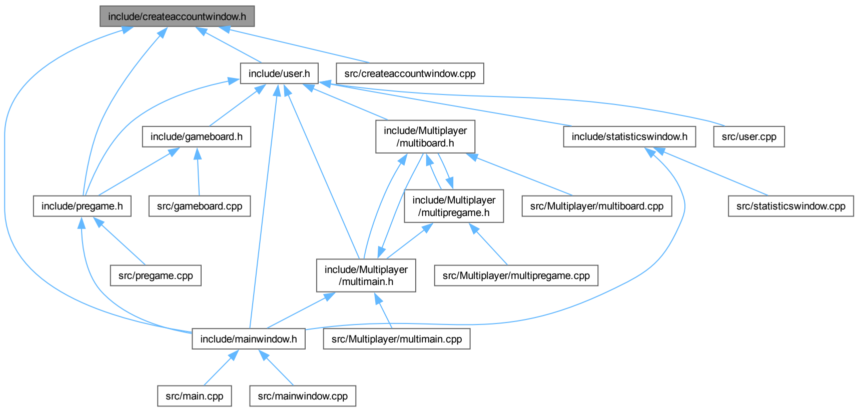
```
#include <QDir>
#include <QFile>
#include <QHBoxLayout>
#include <QJsonDocument>
#include <QJsonObject>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QVBoxLayout>
```

`#include <QWidget>`

Include dependency graph for createaccountwindow.h:



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


include/createaccountwindow.h
include/user.h
src/createaccountwindow.cpp
include/gameboard.h
include/Multiplayer/multiboard.h
include/statisticswindow.h
src/user.cpp
include/pregame.h
src/gameboard.cpp
include/Multiplayer/multipregame.h
src/Multiplayer/multiboard.cpp
src/statisticswindow.cpp
src/pregame.cpp
include/Multiplayer/multimain.h
src/Multiplayer/multipregame.cpp
include/mainwindow.h
src/Multiplayer/multimain.cpp
src/main.cpp
src/mainwindow.cpp

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.

```
#ifndef CREATEACCOUNTWINDOW_H
#define CREATEACCOUNTWINDOW_H

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

class CreateAccountWindow : public QWidget {
  Q_OBJECT

 public:
  static CreateAccountWindow* getInstance(QWidget* parent = nullptr);

  void setPreviousScreen(QWidget* previous);

 public slots:
  void show();

 private:
  explicit CreateAccountWindow(QWidget* parent = nullptr);

  static CreateAccountWindow* instance;

 private slots:
  void onCreateAccountClicked();

  void goBack();

 signals:
  void back();

  void accountCreated();

 private:
  void saveJsonFile(const QString& username);

  QLineEdit* usernameEdit;

  QPushButton* createAccountButton;

  QLabel* statusLabel;

  QString jsonFilePath = "resources/profile.json";  // Update path as necessary

  QWidget* previousScreen = nullptr;
};

#endif  // CREATEACCOUNTWINDOW_H
```

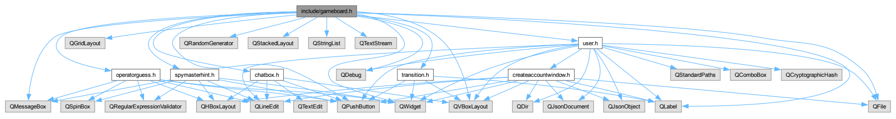
5.5 include/gameboard.h File Reference

Header file for the GameBoard class, which implements a game board for the Spy Master game.

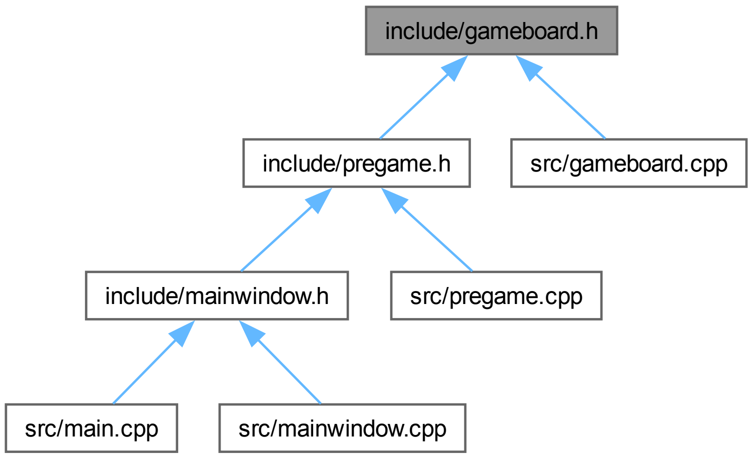
```
#include <QDebug>
#include <QFile>
#include <QGridLayout>
#include <QLabel>
#include <QMessageBox>
#include <QPushButton>
#include <QRandomGenerator>
#include <QStackedLayout>
```

```
#include <QStringList>
#include <QTextStream>
#include <QVBoxLayout>
#include <QWidget>
#include "chatbox.h"
#include "operatorguess.h"
#include "spymasterhint.h"
#include "transition.h"
#include "user.h"
```

Include dependency graph for gameboard.h:



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



Classes

- class GameBoard

  A class representing the game board for the Spy Master game.
- struct GameBoard::Card

  Structure representing a card in the game grid.

5.5.1 Detailed Description

Header file for the GameBoard class, which implements a game board for the Spy Master game.

Author

Group 9

5.6 gameboard.h

Go to the documentation of this file.

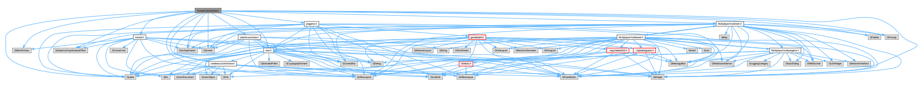
```cpp


#ifndef GAMEBOARD_H
#define GAMEBOARD_H

#include <QDebug>
#include <QFile>
#include <QGridLayout>
#include <QLabel>
#include <QMessageBox>
#include <QPushButton>
#include <QRandomGenerator>
#include <QStackedLayout>
#include <QStringList>
#include <QTextStream>
#include <QVBoxLayout>
#include <QWidget>

#include "chatbox.h"
#include "operatorguess.h"
#include "spymasterhint.h"
#include "transition.h"
#include "user.h"


class GameBoard : public QWidget {
  Q_OBJECT

 public:
  explicit GameBoard(const QString& redSpyMaster, const QString& redOperative,
                     const QString& blueSpyMaster, const QString& blueOperative,
                     QWidget* parent = nullptr);

  ~GameBoard();

  void setRedSpyMasterName(const QString& name);

  void setRedOperativeName(const QString& name);

  void setBlueSpyMasterName(const QString& name);

  void setBlueOperativeName(const QString& name);

  void updateTeamLabels();

 signals:
  void gameEnded();

 public slots:
  void show();

  void displayHint(const QString& hint, int number);

  void displayGuess();

 private:
  void loadWordsFromFile();

  void generateGameGrid();

  void setupUI();

  void nextTurn();

  void onCardClicked(int row, int col);

  void onContinueClicked();

  void showTransition();

  void updateScores();

  void checkGameEnd();

  void endGame(const QString& message);

  void resetGame();

  enum CardType { RED_TEAM, BLUE_TEAM, NEUTRAL, ASSASSIN };

  enum Turn { RED_SPY, RED_OP, BLUE_SPY, BLUE_OP };

  struct Card {
    QString word;
    CardType type;
    bool revealed;
  };

  int currentTurn;
  int redCardsRemaining;
  int blueCardsRemaining;

  int maxGuesses = 0;
  int currentGuesses = 0;

  QString redSpyMasterName;
  QString redOperativeName;
  QString blueSpyMasterName;
  QString blueOperativeName;

  static const int GRID_SIZE = 5;
  Card gameGrid[GRID_SIZE][GRID_SIZE];
  QStringList wordList;

  QGridLayout* gridLayout;
  QPushButton* cards[GRID_SIZE][GRID_SIZE];

  QLabel* redTeamLabel;
  QLabel* blueTeamLabel;
  QLabel* currentTurnLabel;

  SpymasterHint* spymasterHint;
  OperatorGuess* operatorGuess;
  QLabel* currentHint;
  QString correspondingNumber;

  Transition* transition;

  QLabel* redScoreLabel;
  QLabel* blueScoreLabel;

  ChatBox* chatBox;
  QString currentPlayerName;
  ChatBox::Team currentPlayerTeam;
  User* users;
};

#endif  // GAMEBOARD_H
```

5.7 include/mainwindow.h File Reference

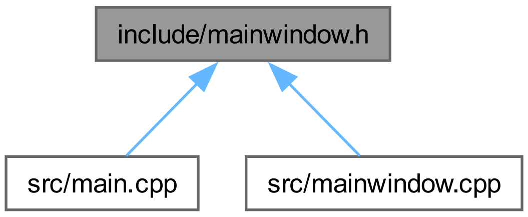
Declaration of the MainWindow class.

```
#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.

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#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"

class PreGame;
class User;
class CreateAccountWindow;
class StatisticsWindow;
class Tutorial;
class MultiMain;

class MainWindow : public QMainWindow {
  Q_OBJECT

 public:
  explicit MainWindow(QWidget* parent = nullptr);

  ~MainWindow();

 public slots:
  void showMainWindow();

 private slots:
  void openPreGame();

  void openOnlineGame();

  void openStatsWindow();

  void openCreateAccount();

  void openTutorial();

  void openMultiMain();

 private:
  QWidget* centralWidget;
  QVBoxLayout* layout;

  QLabel* titleLabel;

  PreGame* preGameWindow;
  MultiMain* multiMain;
  QPushButton* localPlayButton;
  QPushButton* onlinePlayButton;
  QPushButton* tutorialButton;
  QPushButton* statsButton;
  QPushButton*
      createAccountButton;

  User* onlineGameWindow;
  CreateAccountWindow*
      createAccountWindow;
  StatisticsWindow*
      statsWindow;
  Tutorial* tutorialWindow;
};

#endif  // MAINWINDOW_H
```

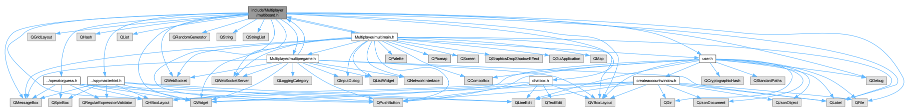
5.9 include/Multiplayer/multiboard.h File Reference

Header file for the MultiBoard class, which implements a multiplayer game board.

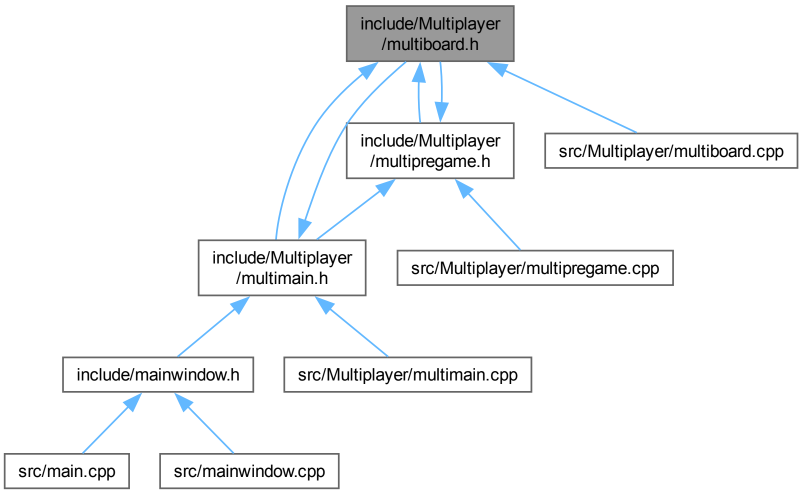
```
#include <QDebug>
#include <QFile>
#include <QGridLayout>
#include <QHBoxLayout>
#include <QHash>
#include <QLabel>
#include <QList>
#include <QMessageBox>
#include <QPushButton>
#include <QRandomGenerator>
#include <QString>
#include <QStringList>
#include <QVBoxLayout>
#include <QWebSocket>
#include <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.

```cpp

#ifndef MULTIBOARD_H
#define MULTIBOARD_H

#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"

class MultiMain;
class MultiPregame;

class MultiBoard : public QWidget {
  Q_OBJECT

 public:
  explicit MultiBoard(bool isHost, QWebSocketServer* server,
                      QList<QWebSocket*> clients, QWebSocket* clientSocket,
                      const QHash<QString, QString>& playerRoles,
                      const QString& currentUsername,
                      QWidget* parent = nullptr  //,
                      //  MultiPregame* pregame = nullptr
  );

  enum CardType {
    RED_TEAM,
    BLUE_TEAM,
    NEUTRAL,
    ASSASSIN
  };

  enum Turn {
    RED_SPY,
    RED_OP,
    BLUE_SPY,
    BLUE_OP
  };

  struct Card {
    QString word;
    CardType type;
    bool revealed;
  };

 public slots:
  void handleTileClick();

  void processMessage(const QString& message);

  void socketDisconnected();

  void handleNewConnection();

 signals:
  void goBack();

 private:
  // Network setup

  bool m_isHost;
  QWebSocketServer* m_server;
  QList<QWebSocket*> m_clients;
```

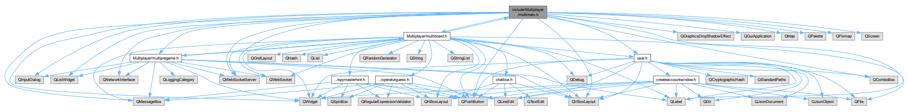
```
    QWebSocket* m_clientSocket;
    MultiPregame* m_pregame;

    ChatBox* chatBox;

    User* users;

    MultiMain* main;

    // Player information
    QHash<QString, QString> m_playerRoles;
    QString m_currentUsername;
    QString m_currentRole;

    // Game board components
    QVBoxLayout* gameVerticalLayout;
    QHBoxLayout* mainLayout;
    QGridLayout* m_grid;
    QLabel* m_playerInfoLabel;
    QLabel* m_turnLabel;
    QList<QPushButton*> m_tiles;
    SpymasterHint* hint;
    OperatorGuess* guess;

    QLabel* blueCardText;
    QLabel* redCardText;

    // Game state
    QStringList m_words;
    QStringList m_tileColors;
    QStringList m_turnOrder;
    int m_currentTurnIndex;

    // Cards remaining
    int redCardsRemaining;
    int blueCardsRemaining;

    void setupUI();

    void setupBoard();

    void initializeWords();

    void initializeBoardColors();

    void sendInitialGameState();

    void loadWordsFromFile();

    void generateGameGrid();

    void checkGameEnd();

    void processChatMessage(const QString& playerName, const QString& message);

    void revealTile(int row, int col, bool broadcast = true);

    void advanceTurn();

    void advanceTurnSpymaster(const QString& hint, int number);

    void updateTurnDisplay();

    void sendToAll(const QString& message);

    void displayHint(const QString& hint, int number);

    void endGame(const QString& message);

    static const int GRID_SIZE = 5;
    Card gameGrid[GRID_SIZE][GRID_SIZE];
    QStringList wordList;
    QPushButton* cards[GRID_SIZE][GRID_SIZE];
    QLabel* currentHint;
    QString correspondingNumber;

    bool isMyTurn() const;

    QString getMyTeam() const;

    QString getColorStyle(const QString& color) const;
};

#endif  // MULTIBOARD_H
```

5.11 include/Multiplayer/multimain.h File Reference

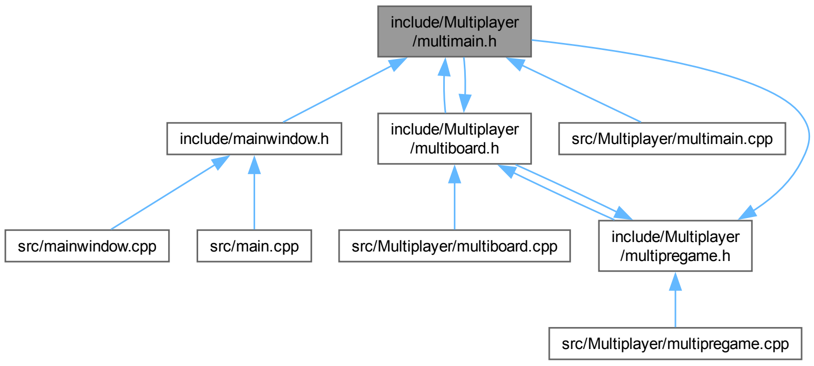
Header file for the MultiMain class, which implements the main multiplayer game lobby interface.

```
#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.

```
#ifndef MULTIMAIN_H
#define MULTIMAIN_H

#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"

class MultiMain : public QWidget {
  Q_OBJECT
 public:
  explicit MultiMain(QWidget* parent = nullptr);

  ~MultiMain();

  void showMainWindow();

 signals:
  void backToMainWindow();

  void enterPregameAsHost(QWebSocketServer* server, const QString& username);

  void enterPregameAsClient(QWebSocket* socket, const QString& username);

 private slots:
  void openMainWindow();

  void onCreateRoomClicked();

  void onJoinRoomClicked();

  void onNewConnection();

  void processTextMessage(QString message);

  void socketDisconnected();

  void onConnected();

  void onDisconnected();

 private:
  // Network members
  QWebSocketServer* m_server = nullptr;
  QWebSocket* m_clientSocket = nullptr;
  QList<QWebSocket*> m_clients;
  QMap<QWebSocket*, QString> m_usernames;
```

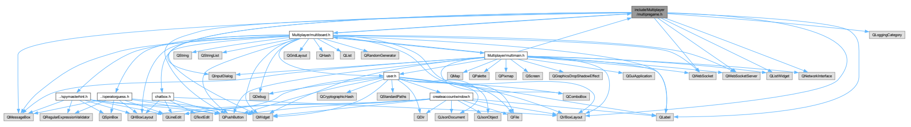
```
    QString m_username;

    // UI members
    QLabel* titleLabel;
    QPushButton* createRoomButton;
    QPushButton* joinRoomButton;
    QPushButton* backButton;

    void updateLobbyList();

    void sendLobbyListToAll();
};

#endif  // MULTIMAIN_H
```

5.13 include/Multiplayer/multipregame.h File Reference

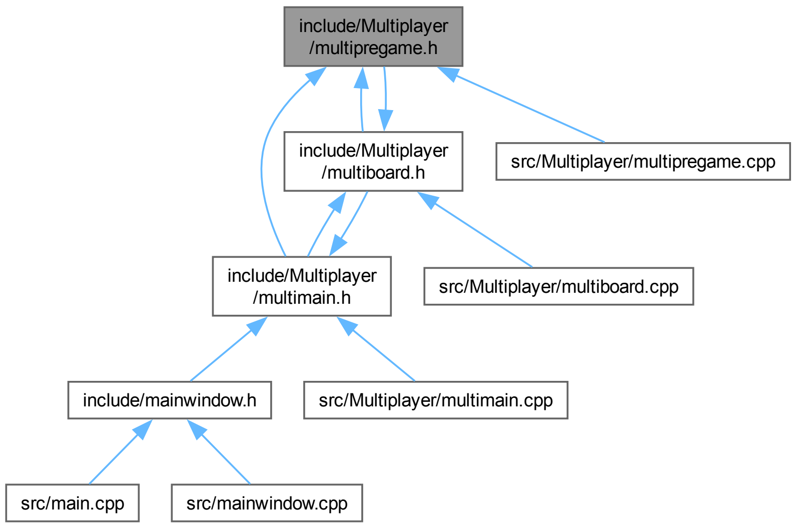
Header file for the MultiPregame class, which implements the pre-game lobby for multiplayer games.

```
#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.

```
#ifndef MULTIPREGAME_H
#define MULTIPREGAME_H

#include <QInputDialog>
#include <QLabel>
#include <QListWidget>
#include <QLoggingCategory>
#include <QMessageBox>
#include <QNetworkInterface>
#include <QPushButton>
#include <QVBoxLayout>
#include <QWebSocket>
#include <QWebSocketServer>
#include <QWidget>

```

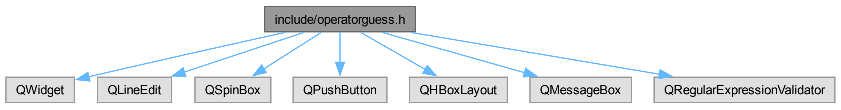
```
#include "Multiplayer/multiboard.h"

class MultiPregame : public QWidget {
  Q_OBJECT
 public:
  MultiPregame(QWebSocketServer* server, const QString& username,
               QWidget* parent = nullptr);

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

  ~MultiPregame();

  void clearUI();

 signals:
  void backToMultiMain();

  void enterPregameAsHost(QWebSocketServer* server, const QString& username);

  void enterPregameAsClient(QWebSocket* socket, const QString& username);

 public slots:
  void onNewConnection();

  void processMessage(const QString& message);

  void socketDisconnected();

  void startGame();

 private:
  void resetUIState();

  void setupUI();

  void sendLobbyUpdate();

  void handleRoleSelection(const QString& message, QWebSocket* sender);

  void gameStarted(bool isHost, QWebSocketServer* server,
                   const QList<QWebSocket*>& clients, QWebSocket* clientSocket,
                   const QHash<QString, QString>& playerRoles);

  void showPregame();

  QWebSocketServer* m_server = nullptr;
  QWebSocket* m_clientSocket = nullptr;
  QList<QWebSocket*> m_clients;
  QMap<QWebSocket*, QString> m_usernames;
  QMap<QWebSocket*, QString> m_roles;
  QMap<QWebSocket*, bool> m_checked;

  QListWidget* playerList;
  QString m_username;
  bool m_isHost;
};

#endif  // MULTIPREGAME_H
```

5.15 include/operatorguess.h File Reference

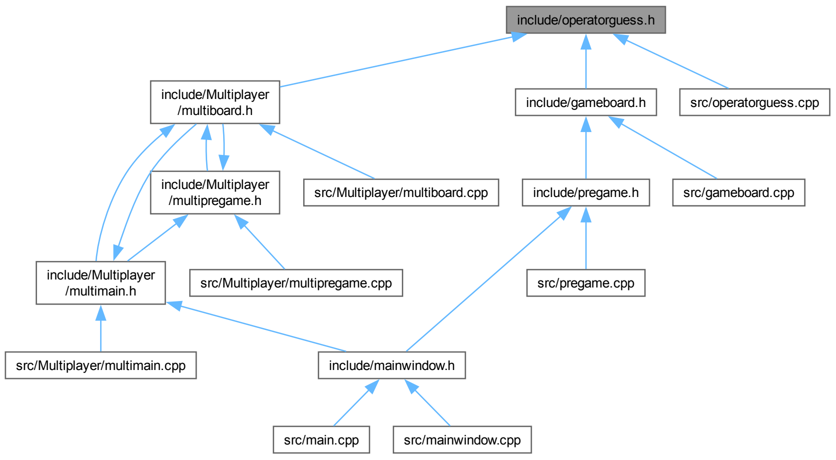
Header file for the OperatorGuess class, which handles operator guessing interface.

```
#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:


include/operatorguess.h
include/Multiplayer/multiboard.h
include/gameboard.h
src/operatorguess.cpp
include/Multiplayer/multipregame.h
src/Multiplayer/multiboard.cpp
include/pregame.h
src/gameboard.cpp
include/Multiplayer/multimain.h
src/Multiplayer/multipregame.cpp
src/pregame.cpp
src/Multiplayer/multimain.cpp
include/mainwindow.h
src/main.cpp
src/mainwindow.cpp

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
```

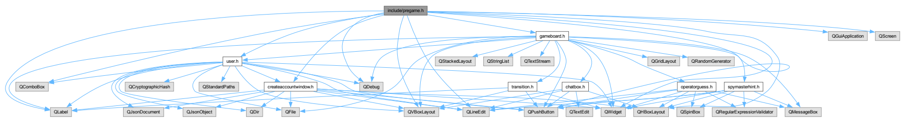
```
#include <QWidget>
#include <QLineEdit>
#include <QSpinBox>
#include <QPushButton>
#include <QHBoxLayout>
#include <QMessageBox>
#include <QRegularExpressionValidator>

class OperatorGuess : public QWidget {
    Q_OBJECT

public:
    explicit OperatorGuess(QWidget* parent = nullptr);

    ~OperatorGuess();

    void reset();

signals:
    void guessSubmitted();

private slots:
    void submitGuess();

private:

    QPushButton* submitGuessButton;
};

#endif // OPERATORGUESS_H
```

5.17 include/pregame.h File Reference

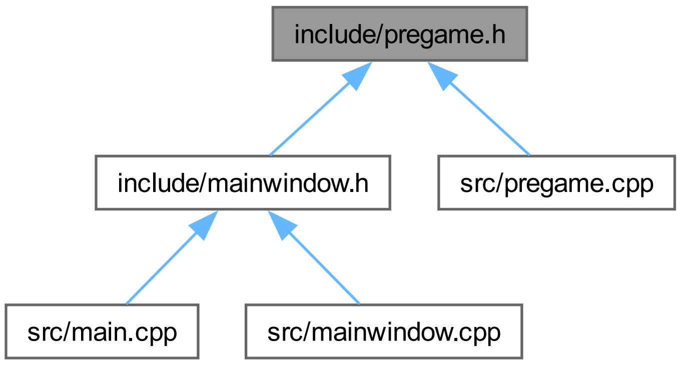
Header file for the PreGame class which handles the game setup screen.

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

Include dependency graph for pregame.h:



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



Classes

- class PreGame

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

5.17.1 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.

```

#ifndef PREGAME_H
#define PREGAME_H

#include <QComboBox>
#include <QDebug>
#include <QGuiApplication>
#include <QHBoxLayout>
#include <QLabel>
#include <QLineEdit>
#include <QPushButton>
#include <QScreen>
#include <QVBoxLayout>
#include <QWidget>

#include "createaccountwindow.h"
#include "gameboard.h"
#include "user.h"

class User;
class CreateAccountWindow;

class PreGame : public QWidget {
  Q_OBJECT

 public:
  explicit PreGame(QWidget* parent = nullptr);

  ~PreGame();

  QString getRedTeamSpyMasterNickname() const;

  QString getRedTeamOperativeNickname() const;

  QString getBlueTeamSpyMasterNickname() const;

  QString getBlueTeamOperativeNickname() const;

 public slots:
  void show();

 private:
  void populateUserDropdowns();

 private slots:
  void goBackToMain();

  void startGame();

  void handleGameEnd();

  void openCreateAccount();

 signals:
  void backToMainWindow();

  void start();

  void update();

 private:
  User* users;

  QStringList usernames;

  CreateAccountWindow* createAccountWindow;

  QLabel* label;

  QPushButton* backButton;

  QPushButton* createAccountButton;

  QPushButton* startButton;

  QComboBox* redTeamSpyMasterComboBox;

  QComboBox* redTeamOperativeComboBox;

  QComboBox* blueTeamSpyMasterComboBox;

```

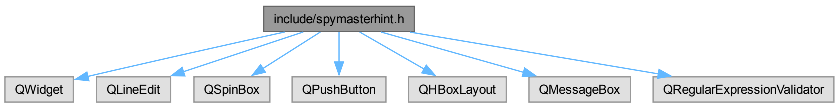
```
    QComboBox* blueTeamOperativeComboBox;

    QVBoxLayout* layout;

    QHBoxLayout* teamsLayout;

    QVBoxLayout* redTeamLayout;

    QVBoxLayout* blueTeamLayout;

    QHBoxLayout* buttonsLayout;

    GameBoard* gameBoard;
};

#endif  // PREGAME_H
```

5.19 include/spymasterhint.h File Reference

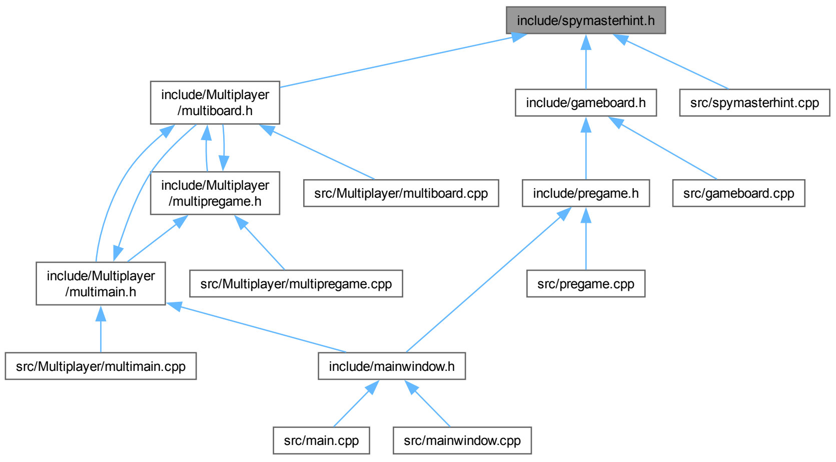
Header file for the SpymasterHint class, which provides a UI for the spymaster to give hints.

```
#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:


include/spymasterhint.h
include/Multiplayer /multiboard.h
include/gameboard.h
src/spymasterhint.cpp
include/Multiplayer /multipregame.h
src/Multiplayer/multiboard.cpp
include/pregame.h
src/gameboard.cpp
include/Multiplayer /multimain.h
src/Multiplayer/multipregame.cpp
src/pregame.cpp
src/Multiplayer/multimain.cpp
include/mainwindow.h
src/main.cpp
src/mainwindow.cpp

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.

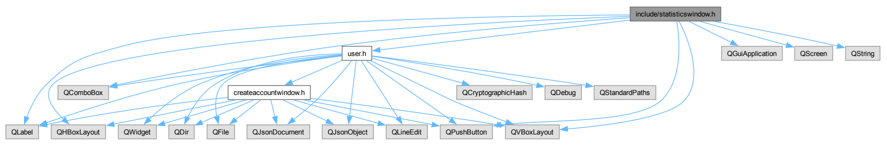
```
#ifndef SPYMASTERHINT_H
#define SPYMASTERHINT_H

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

class SpymasterHint : public QWidget {
  Q_OBJECT

public:
  explicit SpymasterHint(QWidget* parent = nullptr);

  ~SpymasterHint();

  void reset();

signals:
  void hintSubmitted(const QString& hint, const int number);

private slots:
  void submitHint();

  void updateButtonClickable();

  void textToUppercase(const QString& text);

private:
  QLineEdit* hintLineEdit;

  QSpinBox* numberSpinBox;

  QPushButton* giveClueButton;

  QRegularExpressionValidator* textValidator;
};

#endif // SPYMASTERHINT_H
```

5.21 include/statisticswindow.h File Reference

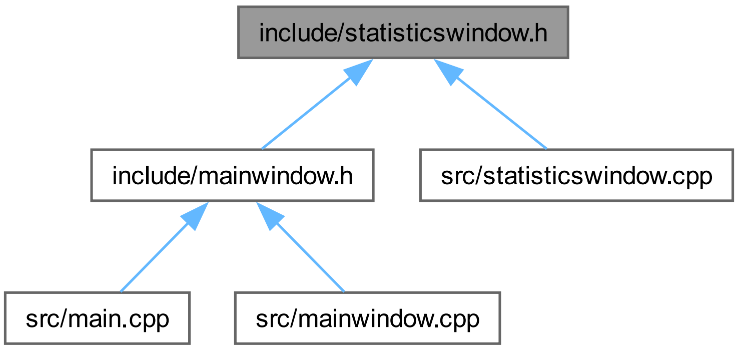
The screen to show the user's statistics.

```
#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.

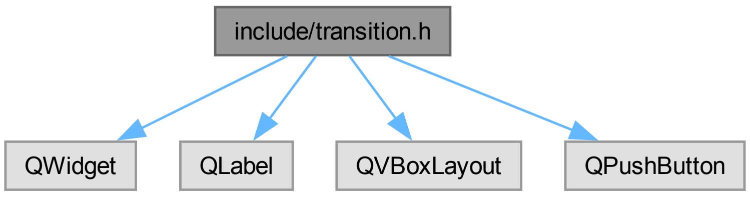
```

#ifndef STATISTICS_WINDOW_H
#define STATISTICS_WINDOW_H

// Qt framework includes for UI components and screen management
#include <QComboBox>        // For dropdown menu of usernames
#include <QGuiApplication>  // For application-level GUI functionality
#include <QHBoxLayout>      // For horizontal layout arrangement
#include <QLabel>           // For text display in UI
#include <QPushButton>      // For button UI elements
#include <QScreen>          // For screen geometry information
#include <QString>          // For string handling
#include <QVBoxLayout>      // For vertical layout arrangement

#include "user.h"  // Include for user data access

// Forward declaration to resolve circular dependency
class User;

class StatisticsWindow : public QWidget {
  Q_OBJECT  // Qt macro for enabling signals and slots mechanism

      signals :
      void
      backToMainWindow();

 public:
  explicit StatisticsWindow(QWidget* parent = nullptr);

  ~StatisticsWindow();

 public slots:
  void show();

 private:
  User* users;

  QPushButton* backToMainButton;

  QComboBox* usernameComboBox;

  QPushButton* showUserStatsButton;

  QString username;

  QLabel* usernameTitle;

  QLabel* gamesPlayedStats;

  QLabel* gamesWinStats;

  QLabel* gamesWinRateStats;

  QLabel* guessTotalStats;

  QLabel* guessHitStats;

  QLabel* guessHitRateStats;

 private:
  void populateDropDown();

 private slots:
  void goBackToMain();

  void showUserStats();
};

#endif  // STATISTICS_WINDOW_H
```

5.23 include/transition.h File Reference

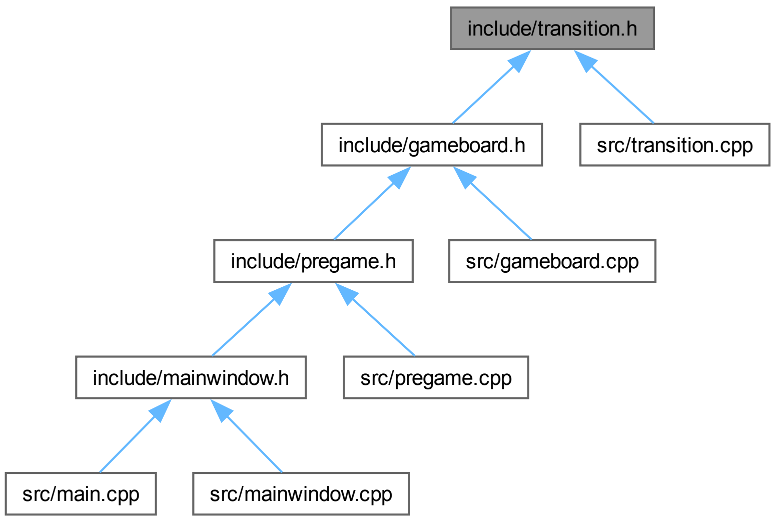
Header file for the Transition class, which provides a UI for transitions between game states.

```
#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.

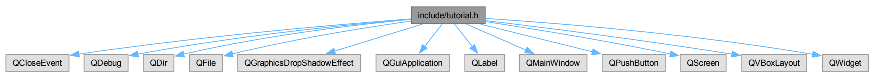
```

#ifndef TRANSITION_H
#define TRANSITION_H

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

class Transition : public QWidget {
    Q_OBJECT

public:
    explicit Transition(QWidget* parent = nullptr);

    ~Transition();

    void setMessage(const QString& message);

signals:
    void continueClicked();

private:
    QLabel* messageLabel;

    QPushButton* continueButton;
};

#endif // TRANSITION_H
```

5.25 include/tutorial.h File Reference

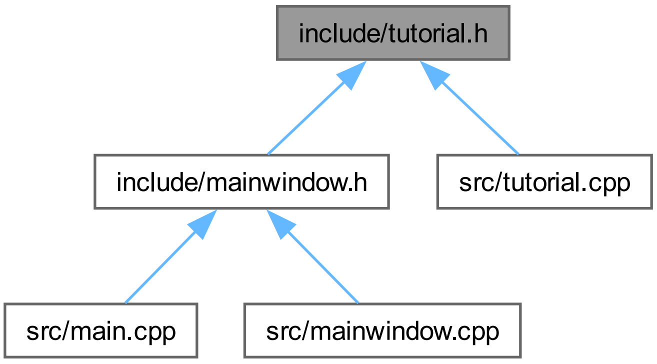
Declaration of the Tutorial class.

```
#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.

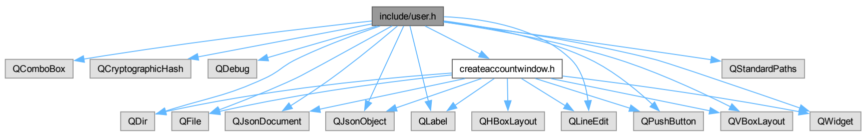
```
#ifndef TUTORIAL_H
#define TUTORIAL_H

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

class Tutorial : public QMainWindow {
  Q_OBJECT

 public:
  explicit Tutorial(QWidget* parent = nullptr);

  ~Tutorial();

 signals:
  void tutorialClosed();

 protected:
  void closeEvent(QCloseEvent* event) override;

 private slots:
  void onContinueClicked();

 private:
  void updateContinueButtonPosition();

  void resetTutorial();

  QWidget* centralWidget;
  QLabel* titleLabel;
  QLabel* textBox;
  QPushButton* continueButton;
  int clickCount;
};

#endif // TUTORIAL_H
```

5.27 include/user.h File Reference

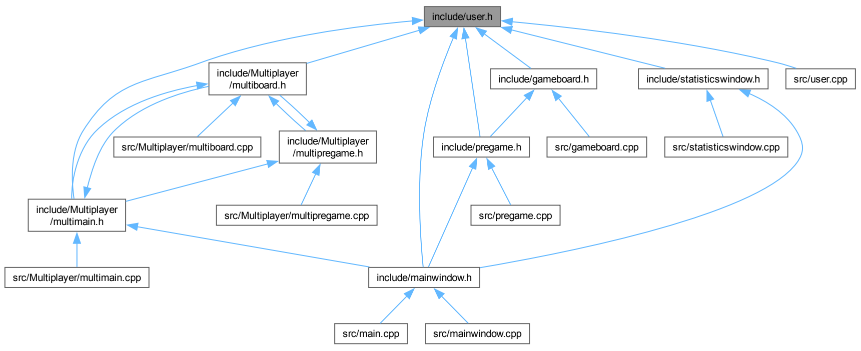
User class to handle local log in and loading/storing json files.

```
#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:


include/user.h
include/Multiplayer/multiboard.h
include/gameboard.h
include/statisticswindow.h
src/user.cpp
src/Multiplayer/multiboard.cpp
include/Multiplayer/multipregame.h
include/pregame.h
src/gameboard.cpp
src/statisticswindow.cpp
include/Multiplayer/multimain.h
src/Multiplayer/multipregame.cpp
src/pregame.cpp
src/Multiplayer/multimain.cpp
include/mainwindow.h
src/main.cpp
src/mainwindow.cpp

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

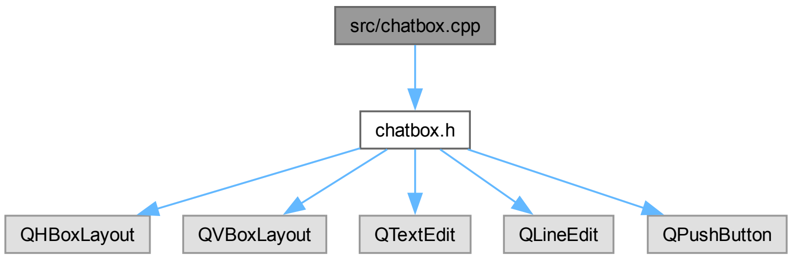
Go to the documentation of this file.

```
#ifndef USER_H
#define USER_H

// Qt framework includes for UI components and file handling
#include <QComboBox>           // For dropdown menu of usernames
#include <QCryptographicHash>  // For password hashing functionality
#include <QDebug>              // For debug output to console
#include <QDir>                // For directory manipulation
#include <QFile>               // For file I/O operations
#include <QJsonDocument>       // For JSON document parsing
#include <QJsonObject>         // For JSON object manipulation
#include <QLabel>              // For text display in UI
#include <QLineEdit>           // For text input fields
#include <QPushButton>         // For button UI elements
#include <QStandardPaths>      // For accessing standard file locations
#include <QVBoxLayout>         // For vertical layout arrangement
#include <QWidget>             // Base class for all UI elements

#include "createaccountwindow.h"  // Include for account creation UI

// Forward declaration to resolve circular dependency
class CreateAccountWindow;

class User : public QWidget {
 Q_OBJECT  // Qt macro for enabling signals and slots mechanism

     public :
     static User*
     instance(QWidget* parent = nullptr);

  ~User();

  void updateGamesPlayed(const QString& username,
                         const unsigned int& newGamesPlayed);

  unsigned int getGamesPlayed(const QString& username) const;

  void updateWins(const QString& username, const unsigned int& newWins);

  unsigned int getWins(const QString& username) const;

  float getWinRate(const QString& username) const;

  void updateGuessTotal(const QString& username,
                        const unsigned int& newGuessTotal);

  unsigned int getGuessTotal(const QString& username) const;

  void updateGuessHit(const QString& username, const unsigned int& newGuessHit);

  unsigned int getGuessHit(const QString& username) const;

  float getHitRate(const QString& username);

  void renameUser(const QString& oldUsername, const QString& newUsername);

  void won(const QString& username);

  void lost(const QString& username);

  void hit(const QString& username);

  void miss(const QString& username);

  QJsonObject loadJsonFile();  // Function to load JSON data

 public slots:
  void show();

 signals:
  void backToMainMenu();

 private slots:
  void handleLogin();

  void refreshUserDropdown();

  void handleCreateAccount();

  void showMainMenu();

 private:
  explicit User(QWidget* parent = nullptr);

  CreateAccountWindow* createAccountWindow;

  QString jsonFilePath = "resources/profile.json";

  QPushButton* backButton;

  QPushButton* createAccountButton;

  QComboBox* usernameComboBox;

  QLabel* jsonContentLabel;

  QPushButton* loginButton;

  void populateUsernameComboBox(const QJsonObject& jsonObject);
};

#endif  // USER_H
```

5.29 src/chatbox.cpp File Reference

`#include "chatbox.h"`

Include dependency graph for chatbox.cpp:

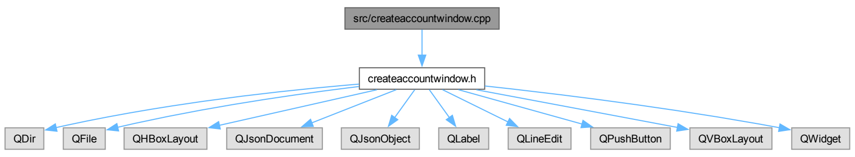


5.30 src/createaccountwindow.cpp File Reference

CPP file for the CreateAccountWindow class which handles user account creation.

`#include "createaccountwindow.h"`

Include dependency graph for createaccountwindow.cpp:



5.30.1 Detailed Description

CPP file for the CreateAccountWindow class which handles user account creation.

Author

Team 9 - UWO CS 3307

Version

0.1

Date

2025-03-30

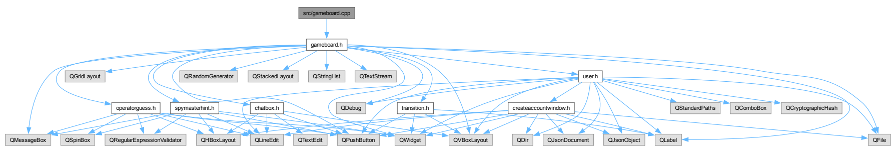
Copyright

Copyright (c) 2025

5.31 src/gameboard.cpp File Reference

```
#include "gameboard.h"
```

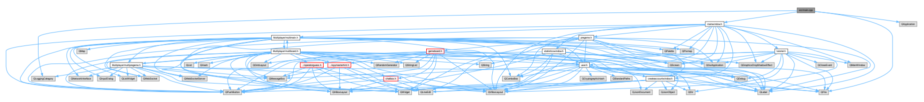
Include dependency graph for gameboard.cpp:



5.32 src/main.cpp File Reference

```
#include "mainwindow.h"
#include <QFile>
#include <QApplication>
#include <QLoggingCategory>
```

Include dependency graph for main.cpp:



Functions

- int main (int argc, char ∗argv[ ])

5.32.1 Function Documentation

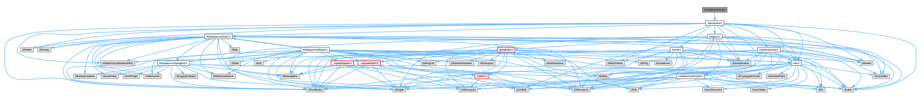
5.32.1.1 main()

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

5.33 src/mainwindow.cpp File Reference

```
#include "mainwindow.h"
```

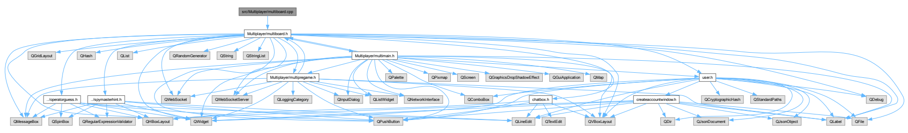
Include dependency graph for mainwindow.cpp:



5.34 src/Multiplayer/multiboard.cpp File Reference

```
#include "Multiplayer/multiboard.h"
```

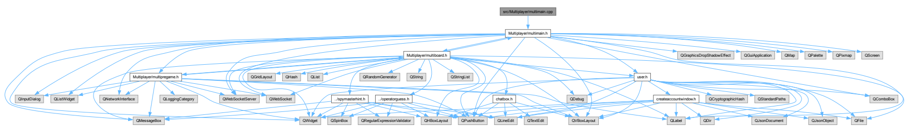
Include dependency graph for multiboard.cpp:



5.35 src/Multiplayer/multimain.cpp File Reference

```
#include "Multiplayer/multimain.h"
```

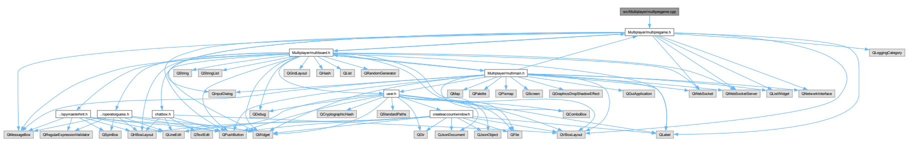
Include dependency graph for multimain.cpp:



5.36 src/Multiplayer/multipregame.cpp File Reference

```
#include "Multiplayer/multipregame.h"
```

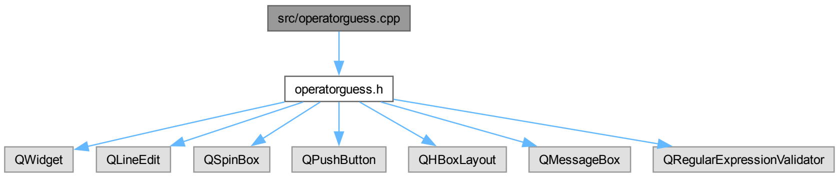
Include dependency graph for multipregame.cpp:



5.37 src/operatorguess.cpp File Reference

```
#include "operatorguess.h"
```

Include dependency graph for operatorguess.cpp:

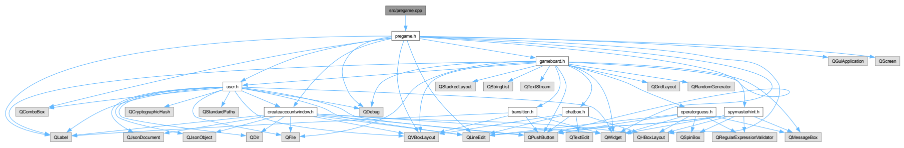


5.38 src/pregame.cpp File Reference

CPP file for the PreGame class which handles the game setup screen.

```
#include "pregame.h"
```

Include dependency graph for pregame.cpp:



5.38.1 Detailed Description

CPP file for the PreGame class which handles the game setup screen.

Author

Team 9 - UWO CS 3307

Version

0.1

Date

2025-03-30

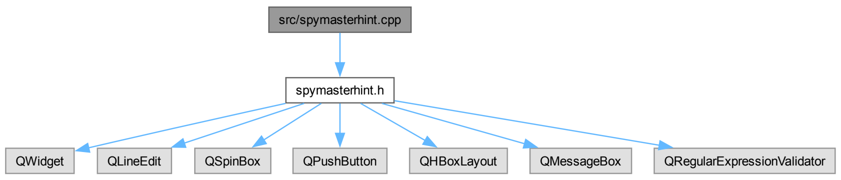
Copyright

Copyright (c) 2025

5.39 src/spymasterhint.cpp File Reference

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

Include dependency graph for spymasterhint.cpp:

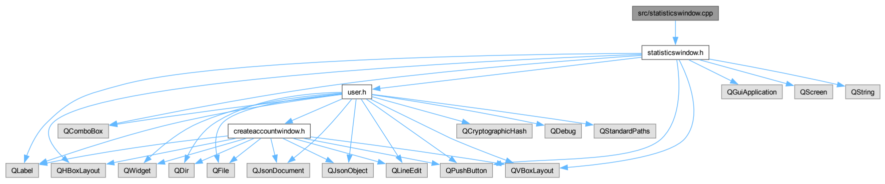


5.40 src/statisticswindow.cpp File Reference

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

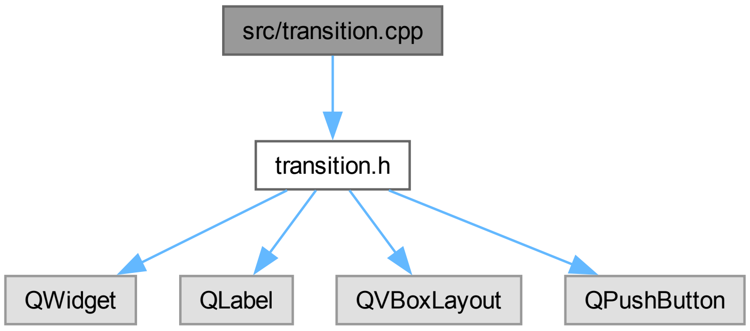
Copyright

Copyright (c) 2025

5.41 src/transition.cpp File Reference

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

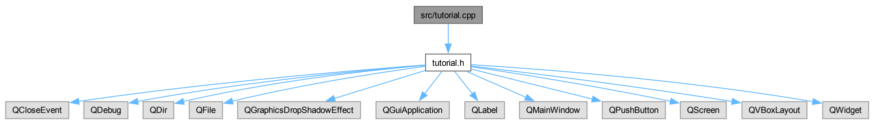
Include dependency graph for transition.cpp:



5.42 src/tutorial.cpp File Reference

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

Include dependency graph for tutorial.cpp:

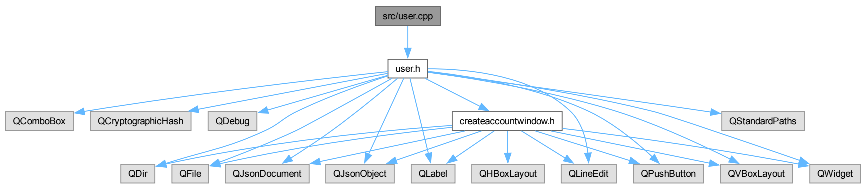


5.43 src/user.cpp File Reference

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