**Operation:** User :: logIn (username: String, password: String)  
**Scope:** Account;  
**Messages:** Client :: { sendLoginInfo(); userValid(); userInvalid\_e() }; Server :: { validateUser(username:String, password:String) }; Screen :: { displayLoginScreen(); displayLoginSuccessScreen(); displayLoginFailureScreen() }  
**Pre:** The user is currently not logged in.  
**Description:** The operation shows the log in form on the screen, and if the entered username and password are validated by the server, the user is logged in while a success message is displayed on the screen. Otherwise an error message is displayed on the screen.

**Operation:** User :: signUp (username: String, password: String)  
**Scope:** Account;  
**Messages:** Client :: { newUserCredentialsValid(); newUserCredentialsInvalid\_e() }; Server :: { validateNewUserCredentials(username:string, password:string) }; Screen :: { displayNewUserSignUpPage(); displayNewUserSuccessPage(); displayNewUserFailurePage() }  
**Pre:** The user is currently not logged in.  
**New:** newAccount: Account  
**Description:** The operation shows the sign up form on the screen, and if the new credentials entered by the client are validated by the server, a new account is created and a success message is displayed on the screen. Otherwise, an error message is displayed on the screen.

**Operation:** User :: reviewStatistics (player: String)  
**Scope:** Account; Statistics;  
**Messages:** Client :: { requestStatistics(player:String); provideStatistics(stats:Statistics) }; Server :: { requestStatistics(player:String) }; Screen :: { displayStatistics(stats:Statistics) }  
**Pre:** The user is currently logged in.  
**New:** gameStats: Statistics  
**Description:** The client requests the statistics from the server and after it is provided, it is displayed on the screen.

**Operation:** User :: matchup (opponent: String)  
**Scope:** Game; Account;  
**Messages:** Client :: { requestPlayers(); challengePlayer(username:String); acceptGame(from:String); rejectGame(from:String); allCurrentPlayers(players:List); askIfPlayerWantsToJoin(to: String); gameRejected(to:String); gameAccepted(to:String) }; Server :: { requestAllCurrentPlayers(); requestMatching(from:String, to: String); playerAcceptsGame(from:String, to:String); playerRejectsGame(from:String, to:String) }; Screen :: { displayAllPlayersPage(players:List); displayGameAcceptedPage();displayGameRejectedPage() }  
**Pre:** The user is currently logged in and does not have any set up games.  
**Description:** The screen shows all the players after requesting the information. The client can request a matching or can accept/reject a request. Result of the matching is displayed on the screen.

**Operation:** User :: setupGame ()  
**Scope:** Game; GameConditions;   
**Messages:** Client :: { sendNewConditions(conditions:GameConditions); sendConfirmationOfConditions(); newGameConditions(conditions:GameConditions); startGame(conditions:Conditions) }; Server :: { submitGameConditions(conditions:GameConditions); submitAgreement() }; Screen :: { displayGameConditions(conditions:GameConditions); acceptGameConditions(); displayNewGame(conditions:Conditions) }  
**Pre:** The user is currently logged in and does not have any set up games.  
**New:** newConditions: GameCondition; newGame: Game  
**Description:** New game conditions are generated and displayed, if all clients agree, a new game is started. Otherwise, new conditions are generated and a new game creation awaits the confirmation for the newly created conditions of all clients.

**Operation:** User :: loadGame (gameName: String)  
**Scope:** Game; GameConditions; Ship; ShipState;  
**Messages:** Client :: { loadGame(name: String); loadGameSuccess(); loadGameFailure\_e() }; Server :: { loadGameState(name: String); }; Screen :: { saveGameSuccessMessage();saveGameFailureMessage(); listLoadableGames(names:List); loadGameSuccessMessage(); loadGameFailureMessage() }  
**Pre:** The user is currently logged in and does not have any set up games. Also the opponent saved in the game has to be logged in.  
**Description:** All the loadable/saved games requested from the server and are displayed on the screen. After the client chooses which game to load, it is loaded.

**Operation:** User :: saveGame (gameName: String)  
**Scope:** Game; GameConditions; Ship; ShipState;  
**Messages:** Client :: { saveGame(name:String); saveGameSuccess(); saveGameFailure\_e() }; Server :: { saveGameState(name:String); }; Screen :: { saveGameSuccessMessage();saveGameFailureMessage(); listLoadableGames(names:List); loadGameSuccessMessage(); loadGameFailureMessage() }  
**Pre:** The user is currently logged in and is enrolled in the game.  
**Description:** The client chooses to save the game displayed on the screen, if successful, a success message is displayed on the screen. Otherwise a failure message is displayed on the screen.

**Operation:** User :: selectShips (ships: List)  
**Scope:** Game; GameConditions; Ship; ShipState;  
**Messages:** Client :: { listSelectableShips(ships:List); shipSelectionValid(); shipSelectionInvalid\_e(); select(ship:Ship); deselect(ship:Ship); submitShipSelections() }; Server :: { validateShipSelection(ships:List) }; Screen :: { displayShipSelectionScreen(ships:List) }  
**Pre:** The user is currently logged in, matched up with another user and the game is already set up.  
**Description:** Selectable ships are displayed on the screen, then the client makes and submits a selection. After that, the server lets the client know if the ship selection is valid or invalid.

**Operation:** User :: placeShips (ships: List)  
**Scope:** Game; GameConditions; Ship; ShipState;  
Messages: Client :: { placeShip(ship:Ship); removeShip(ship:Ship); submitShipPlacement();placementValid(); placementInvalid\_e(); } ; Server :: { alertOfShipPlacement(shipPlacements:List) } ; Screen :: { displayPlaceShipsPage(); displayInvalidPlacementPage(); displayValidPlacementPage() }  
**Pre:** The user is currently logged in, matched up with another user and the game is already set up. The user has also selected some ships.  
**New:** newShip: Ship  
**Description:** After the ship placements page is displayed on the screen, the client places/removes ships and submits the final placement to the server. If the placements are confirmed to be valid by the server, a valid placement page is displayed on the screen; otherwise, an invalid placement page is displayed on the screen.

**Operation:** User :: makeAction (action: InGameAction)  
**Scope:** Game; InGameAction;  
**Messages:** Client :: { makeAction(action: InGameAction); actionValid(); actionInvalid\_e(); playerWin(); playerLose() }; Server :: { submitAction(action:InGameAction) }; Screen :: { displayActionInvalidPage() }  
**Pre:** The user is currently logged in, matched up with another user and the game is already set up.  
**New:** userAction: InGameAction  
**Description:** The user chooses to make an in game action (as displayed in Figure 2.3 in the Use Cases Section). If the action is valid, it is submitted; otherwise, an action invalid page is displayed on the screen. Also, if an action defines end of game, the client is let know of win/lose status.

**Operation:** User :: updateTurn  
**Scope:** Game; GameConditions;  
**Messages:** Screen :: { displayNewGameScreen() }; Client :: { newGameScreen(shipPlacement:List, shipDamage:List) }  
**Pre:** The user is currently logged in, matched up with another user and the game is already set up.  
**Description:** The server sends the client the placement of ships and the damaged ships. This information is then sent to the screen to display a new game screen.