Operation Model

Note: any message from lobby to server that is inalienably coupled with a “client-to-server” message and vice-versa with GUI to client, is placed in the message portion of that client’s client-to-server message.
Any “sessionID” that cannot be matched with returns an “IDNotFound_e” message.
Note: With client -> server operations the player is the sender, server -> client the recipient is the player.

Server Side Operation Model

Operation: ColtExpressServer::login(username: String, password: String)
Scope: Player, Game;
New:
Messages: Game::{requestOAuthToken, OAuthToken_r}, Player::{loginSuccessful, loginUnsuccessful}
Post: This operation contacts the lobby service actor and looks for a user with the username and password. If no such credentials are present then a login unsuccessful message is communicated to the sender. Else a login successful message and the token of the user is returned by the lobby service actor to the Player represented by the sender.

Operation: ColtExpressServer::requestExistingGames(token: String)
Scope: Player, Game;
New:
Messages: Game::{requestExistingGames, listExistingGames_r}**,** Player::{listExistingGames}
Post: The operation *requestExistingGames* contacts the lobby service actor and receives the list of existing game sessions.

Operation: ColtExpressServer::createNewGame(token: String, username: String, saveGameID: String)
Scope: GameBoard, horse, trainCar, stagecoach, marshal, loot, hostage, angry shotgun, card;
New: newGame: Game; newGameBoard: GameBoard; newHorses: Set{horse}; newLoots: Set{loot}, newCards: Set{Card}, newStagecoach: stagecoach, newMarshal: marshal, newShotgun: AngryShotgun, newTrainCars: Set{TrainCar}
Messages: Game:{createNewGame, newGameCreated_r}, Player{currentGameBoard}
Post: The *createNewGame* operation creates a new Game session using the Lobby Service, creating and initializing all objects of the server in accordance with Colt Express rules, it saves the game board and the associated sessionID in the lobby, presents the gameboard to the player represented by the sender.

Operation: ColtExpressServer::joinGame(token: String, sessionID: String, user: String)
Scope: Player, Game;
New:
Messages: Game::{joinGame, gameJoined_r}, Player::{gameJoined}

Post: The *joinGame* operation allows the player represented by the sender to join a game by querying the lobby for a game with the given sessionID, and setting the player represented by the sender to a participant in that game with username user.

Operation: ColtExpressServer::requestSavedGames(token: String)
Scope: Player, Game;
Messages: Game::{requestSavedGames, listSaveGames_r}**,** Player::{listSaveGames}
Post: This message queries the lobby service for its set of saved games and returns a set of session IDs corresponding to the games saved on the lobby service database.

Operation: ColtExpressServer::deleteGameSession(sessionID: String, token: String)
Scope: Game, Player;
New:
Messages: Game::{deleteGameSession, gameDeleted_r}, Player::{gameDeleted}
Post: This operation searches the Game sessions for the sessionID specified and deletes the Game from the lobby database.

Operation: ColtExpressServer::leaveGame(token: String, sessionID: String, user: String)
Scope: Player, Game;
New:
Messages: Game::{removePlayerFromSession, playerRemovedFromSession_r}, Player::{playerRemovedFromSession}
Post: This operation removes a player from the game session corresponding to the given sessionID by decoupling the participant of that game with a username corresponding to user from the game session.

Operation: ColtExpressServer::updateSessionPlayers(sessionID: String)
Scope: Player, Game;
New:
Message: Game::{updateSessionPlayers, listSessionPlayers_r}, Player::{listSessionPlayers}
Post: The *updatesSessionPlayers* operation refreshes the information received about the participants in the game session

Operation: ColtExpressServer::launchGame(sessionID: String, token: String)
Scope: GameBoard, horse, trainCar, stagecoach, marshal, loot, hostage, angry shotgun, card
New: newGame: Game; newGameBoard: GameBoard; newHorses: Set{horse}; newLoots: Set{loot}, newCards: Set{Card}, newStagecoach: stagecoach, newMarshal: marshal, newShotgun: AngryShotgun, newTrainCars: Set{TrainCar}
Message: Game::{launchGame, loadedGameSuccessfully_r}, Player::{startGame}
Post: The *launchGame* operation loads the game from the selected sessionID. The state of the game is re-initialized. This game state is communicated to the sender.

Operation: ColtExpressServer::saveGame(sessionID: String, token: String)
Scope: Player, Game;

New:
Message: Game::{saveGame, gameSaved_r}, Player::{gameSaved}
Post: This operation saves the game board to the lobby database while tying it to the specified sessionID.

Operation: ColtExpressServer::characterPicked(c: Bandit, token: String)
Scope: Player, Bandit;
New: newBandit: bandit, newCards: Set{Card}
Messages:
Post: This operation instantiates the Bandit association in the Player instance representing the sender to c and instantiates the player with the cards that every Player starts with according to the game rules, all such cards are not visible.

Operation: ColtExpressServer::rideOnChoice(choice: boolean, token: String)
Scope: Player, Horse, Traincar;
New:
Message:
Post: This operation allows the client to decide whether in the HorseAttack phase he/she wishes to continue riding onto another Train car. The player begins the HorseAttack phase in a Position Horse with Horse aggregated adjacent to the caboose, if choice is True then the player will “ride on” and the Horse will be placed in an aggregation adjacent to the Traincar next in the ordered set of train cars composing the game board. If the choice is false then the player will have his position switched from the Horse to the Traincar adjacent to the horse, so long as that Traincar is not a locomotive, the Horse is then made adjacent to this new position on the game board.

Operation: ColtExpressServer::drawCards(token: String)
Scope: Player, Card, Deck;
New:
Messages:
Post: This operation allows a player to draw cards from their deck and move these cards into their hand. The instance of the player representing the sender will transfer a randomly generated set of 3 cards from their deck to their hand, these cards will only be visible to their owner.

Operation: ColtExpressServer::playCard(c: Card, token: String)
Scope: Player, Card, ActionStack, Hand;
New:
Messages:
Post: This operation allows the player to play a card in the scheming phase by transferring the card c from the players hand to the games ActionStack.

Operation: ColtExpressServer::playWhiskey(w: Whiskey, token: String)
Scope: Player, Whiskey, Card, ActionStack, Deck;

New:
Messages:
Post: The player representing the sender will find an instance of whiskey of type w in their liquor field, if the whiskey is full the whiskey will be changed to halfFull, if it is half-full the whiskey will be changed to empty. If w is OldWhiskey you play two action cards in a row, if w is a newWhiskey then you draw 3 more cards and then play a card.

Operation: ColtExpressServer::rob(l: Loot, token: String)
Scope: Player, Loot, Whiskey;
New:
Messages:
Post: The player representing the sender will appropriate the loot l into their loot, the loot l will have its position nullified.

Operation: ColtExpressServer::rideAction(p: Position, token: String)
Scope:
New:
Messages:
Post: This operation allows the player represented by the sender to move their position to any floor level position on the gameboard.
If the player's position is - or is adjacent - to a Traincar with adjacent horse set of cardinality non-zero then the player's position is assigned to p. One of the horses in the horse set adjacent to the player's original position is removed from that set and placed in the adjacent set of p.

Operation: ColtExpressServer::move(p: Position, token: String)
Scope: Player, Stagecoach, TrainCar, Locomotive;
New:
Messages:
Post: This operation moves the position of the player representing the sender to a new position p. P must have been adjacent to the players original position. The players position association is changed to p.

Operation: ColtExpressServer::marshal(p: Position, token: String)
Scope: Player, Marshal, TrainCar, locomotive;
New:
Messages:
Post: This operation allows the player represented by the sender to change the position of the marshal to that of p so long as p is a traincar next to the current position of the marshal in the GameBoards ordered traincar set.

Operation: ColtExpressServer::shoot(c: Bandit, token: String)
Scope: Bandit, Player, Card, Deck;
New:
Messages:
Post: This operation allows the player represented by the sender to shoot a Bandit c, by taking a bulletcard from the players Deck and placing it in the player associated with c's deck.

Operation: ColtExpressServer::punch(c: Punchable, p:Position, l: Loot, token: String)
Scope: Player, Punchable, Character
New:
Messages:
Post: This operation allows the player represented by the sender to punch a chosen player in the space. If the chosen bandit has loot then the bandit loses 1 loot token and is moved to the same floor of an adjacent car.

Client Side Operation Model

Note: Output messages to server from the client are the same as above
Player now represents the GUI actor
Note: "The Player" in the below operations is determined by the receiver of the input message from the server, ie the player is the recipient, where in the above the player was the sender.

Operation: ColtExpressClient::readyToPlay()
Scope: player; Game;
New:
Messages:
Post: There are a sufficient number of characters participating in the game, the players are made aware that the gameBoard is set up.

Operation: ColtExpressClient::regenerateBoard()
Scope: Gameboard, TrainCar, Locomotive, horse;
New:
Messages: Player::{displayGameBoard}
Post: The game session representing the sender generates the current gameboard and displays it to the player.

Operation: ColtExpressClient::promptRideOn()
Scope: Player, Game;
New:
Messages: Player::{promptRideOn, rideOnChoice_r}, Game::{rideOnChoice}

Post: The player is presented with the choice to ride on in the horse attack phase. The player makes their choice and the server is notified/updated accordingly.

Operation: ColtExpressClient::initialPosition(p: Position)
Scope: Bandit, Position, Traincar, GameBoard, Game;
New:
Messages:
Post: The operation *initialPosition* represents the starting position of the bandit on the train car, if all other players have dismounted from their horses (ie have chosenOff) during horseAttack.

Operation: ColtExpressClient::pickACharacter(options: Set{Bandit})
Scope: Player, Bandit, Game;
New:
Messages: Player::{pickACharacter}**,** Game::{CharacterPicked};
Post: Once an unchosen character is picked by the client, a bandit will be assigned to the player along with the bandit's unique power.

Operation: ColtExpressClient::currentGameBoard(g: gameBoard)
Scope: GameBoard, TrainCars, Locomotive, Player, Game;
New:
Messages: Player::{displayGameBoard}
Post: The operation *currenGameBoard* exposes the game board to the GUI Player and

Operation: ColtExpressClient::roundStarted(r: Round)
Scope: Round, Turn, Game, Player
New:
Messages: Player::{roundStarted}
Post: In this operation the first player draws the first card on the deck of round cards informing all the players in the game of the type of turns.

Operation: ColtExpressClient::schemingStarted()
Scope: Game, Hand, ActionStack, Deck
New: newHand: Hand
Messages: Player::{schemingStarted, turnType, drawCards_r, playCard_r, playWhiskey_r}
Post: In this operation, each player draws 6 cards from the deck which forms the players hand. The players are notified of the turn type.

Operation: ColtExpressClient::schemingEnded()
Scope: Action
New: newActionStack: ActionStack, newHand: Hand
Messages:
Post: In a clockwise direction each player either plays an action card forming the action stack or draws 3 cards from the deck.

Operation: ColtExpressClient::stealingStarted()
Scope: GameBoard, ActionStack, Player, Game;
New:
Messages: Player::{stealingStarted}
Post: The action cards are resolved one by one in the order they were played. These actions determine the gameboard state along with the loot and bulletcards held by each player.

Operation: ColtExpressClient::stealingEnded()
Scope: ActionStack, Card, Deck
New:
Messages: Player::{stealingEnded}
Post: In this the cards in the action stack are returned to the players to whom they belong. The next turn is started, if no next turn, the event, if no event, the next round, if no next round the game is over, the gunslinger is announced, the winner is announced.

Operation: ColtExpressClient::roundEnded()
Scope:
New:
Messages: Player::{endOfRoundEvent}
Post: In this operation the player is notified of the end of round event.

Operation: ColtExpressClient::yourTurn()
Scope: Hand, ActionStack, Deck, Game;
New:
Messages: Player::{yourTurn}
Post: On a player's turn, if it is the scheming phase they can either play an action card or draw 3 more cards from the deck and add them to the hand, or if it is the stealing phase the player can resolve the action card they placed, or skip their turn.

Operation: ColtExpressClient::promptSchemingAction()
Scope:
New:
Messages: Player::{promptSchemingAction}
Post: The player is prompted to take a scheming action, by being presented with option

Operation: ColtExpressClient::cardTurned(c: Card)
Scope: Card, ActionStack
New: newActionStack: ActionStack
Messages: Player::{cardTurned}
Post: The action cards played during the schemin phase are turned over in the order they were played.

Operation: ColtExpressClient::promptSchemingAction()
Scope: Player;
New:
Messages: Player::{promptSchemingAction}
Post: The player is prompted to take a scheming action.

Operation: ColtExpressClient::promptRob(options: Set{Loot})
Scope: Loot, Bandit, Deck
New:
Messages: Player::{promptRob, rob_r}
Post: The effects of the operation *promptRob* allows the player to take 1 loot token from the space without looking at its value.

Operation: ColtExpressClient::promptRideAction(options: Set{Position})
Scope: horse, locomotive, stageCoach, TrainCar, GameBoard;
New:
Messages: Player::{promptRideAction, rideAction_r}
Post:

Operation: ColtExpressClient::promptMove(options: Set{Position})
Scope: Position, Player, GameBoard, Traincar, Locomotive, stageCoach;
New:
Messages: Player::{promptMove, move_r}
Post: The player is presented with a list of set positions into which they may move their Bandit, the player chooses a position and the bandit is moved on the gameboard.

Operation: ColtExpressClient::promptMarshal(options: Set{Position})
Scope: Position, Marshal, GameBoard, Traincar, locomotive
New:
Messages: Player::{promptMarshal, marshal_r}
Post: The player is asked where of the possible positions on the board they wish to move the marshal. The player selects an option and the marshal is moved to the set position on the gameboard.

Operation: ColtExpressClient::promptShoot(options: Set{Bandit})
Scope: Bandit, Player, Deck, Hand;
New:
Messages: Player::{promptShoot, shoot_r}
Post: A player is asked which of the available options they wish to shoot by removing one of the bullet cards from their deck and placing it in the deck of the chosen opponent. They choose an option and the action is realized with this opponent in the crosshairs.

Operation: ColtExpressClient::promptPunch()
Scope: Punchable, Character, Loot, Bandit, Deck
New:
Messages: Player::{promptPunch, punch_r}
Post: The effects of the operation *promptPunch* asks the player to choose the bandit in the space. The chosen bandit loses 1 loot token if present.

Operation: ColtExpressClient::loginSuccessful()
Scope: Player;
New:
Messages: Player::{loginSuccessful}
Post: This operation notifies the player that the login was successful.

Operation: ColtExpressClient::loginUnsuccessful()
Scope: Player;
New:
Messages: Player::{loginUnsuccessful}
Post: This operation notifies the player that the login was unsuccessful

Operation: ColtExpressClient::listExistingGames(game: Set{Session})
Scope: Player;
New:
Messages: Player::{listExistingGames}
Post: The player is sent a list of the currently available sessions and their corresponding lobby information.

Operation: ColtExpressClient::gameJoined()
Scope: Player;
New:
Messages: Player::{joinGame}
Post: The player is notified that they have joined the game.

Operation: ColtExpressClient::gameDeleted()
Scope: Player;
New:
Messages: Player::{gameDeleted}
Post: The player is notified that the game was deleted.

Operation: ColtExpressClient::playerRemovedFromSession()
Scope: Player;
New:
Messages: Player::{PlayerRemovedFromSession}

Post: If the player is the player removed from the session they are removed from the game, if the player is not the player removed from the session the player is simply notified of the adjustment.

Operation: ColtExpressClient::listSessionPlayers(players: Set{Players})
Scope: Player;
New:
Messages: Player::{listSessionPlayers}
Post: The players involved in the session of the recipient player are revealed to the recipient player.

Operation: ColtExpressClient::startGame()
Scope: Player;
New:
Messages: Player::{startGame}
Post: The player is notified that the game has started.

Operation: ColtExpressClient::gameSaved()
Scope: Player;
New:
Messages: Player::{gameSaved}
Post: The player is notified that the game has been saved.

Operation: ColtExpressClient::listSavedGames(games: Set{saveGame})
Scope: Player;
New:
Messages: Player::{listSavedGames}
Post: A list of the session IDs of the games saved on the lobby database is revealed to the player.

Operation: ColtExpressClient::gunslingerRecipient(c: Bandit)
Scope: Player;
New:
Messages: Player::{gunslingerRecipient}
Post: The player is notified that the bandit c received the gunslinger award.

Operation: ColtExpressClient::gameWinner(c: Bandit)
Scope: Player;
New:
Messages: Player::{gameWinner}
Post: The player is notified that the bandit c won the game of Colt Express.

For the Client -> Server operations see the above Server Side Operation Model.