Ticket to Ride Phase 2

Objectives

1. Fully implement (client and server side):
   - game setup including the selection of 2 or 3 destination cards
   - the Chat feature.
2. In the client side only: Implement all views for displaying game state except the game summary (the last view). Demonstrate that views are updated for any modification to the client model for any part of the game except the game summary (the last view). (The server side of these views will be implemented in the next phase).
3. Understand how to create and document a design.

Patterns

Update the old patterns for the new capabilities.

Requirements

Design

Create and document the architecture of this phase. You should provide UML diagrams for all classes created or modified in Phase 2. Please see the Phase 2 Design Rubric for detail.

As part of the Phase 2 Design, you must also create two sequence diagrams: 1. Depict the whole flow for selecting/discarding Destination Cards at the start of a game. 2. Depict the whole flow for adding to or updating the chat history.

A player's poller will periodically request commands from that server that it has not seen. This will necessitate the development of command queues for each game. Conceptually there are also command queues for each player. You must provide a means to ensure that when a client requests commands that it gets only the ones it has not seen. This includes those that may have previously been lost in transmission.

As part of the Phase 2 Report, each team member should provide a single significant class documented using Javadoc and Design-by-Contract style. The javadocs must include documentation for the public fields (if any), public constructors, and public methods. Public constructors and method definitions must include descriptions, parameter definitions (if any), exception definitions, and pre- and post- conditions. The class description must provide invariants as needed.

Capabilities (These may be adjusted slightly depending on design)

Game setup capabilities:

- Assign a color to each player
- Determine player turn order
- Have the server randomly select 4 train cards for each player.
- For each player, have the server select 3 destination cards for the player, and allow the player to discard 0 or 1 of them (initial destination card selection). This must be fully implemented i.e. the server knows which (if any) card the player discarded and it gets returned to the destination card deck.

Game view capabilities:

- Map/Game view:
  - Show the map including all the cities and the routes between them.
  - For claimed routes, show which player owns the route. Route ownership must be indicated on the map itself (not elsewhere).
- PlayerInfo view:
  - Have some sort of window, drawer, dialog box, or otherwise that shows information about all players in the game. This includes their names, colors, points, the order in which the players take turns, and indicate whose turn it currently is.
  - For this player, also show the specific train cards (i.e. their colors) and destination cards owned (dealt by server at game initialization). For each opponent player, show the number of train cards and destination cards owned.
- Cards and card decks:
  - For the train card deck, show the top 5 cards that are visible. Also show the number of invisible cards in the deck.
  - For the destination card deck, show the number of cards in the deck.
- Demonstrate that your views update dynamically when your client model is modified. This can be done through a variety of ways. One way would be to add a temporary button to your main view. When pressed, this button could call a method on the view's presenter that makes a series of changes to the client model. For each change, the presenter could:

- Call the view to display a toast describing the change (e.g., "Updating player points").
- Make the indicated change to the local client model (*no server interaction is involved here*). This should cause your client model to notify presenters of the model changes, and presenters should update their views appropriately.
- The following items should be demonstrated:
    - Update player points
    - Add/remove train cards for this player
    - Add/remove player destination cards for this player
    - Update the number of train cards for opponent players
    - Update the number of train *cars* for opponent players
    - Update the number of destination cards for opponent players
    - Update the visible (face up) cards in the train card deck
    - Update the number of invisible (face down) cards in train card deck
    - Update the number of cards in destination card deck
    - Add claimed route (for any player). Show this on the map.
    - Add chat message from any player
    - Advance player turn (change the turn indicator so it indicates another player)
- Include pauses as needed to allow the TA to see each change.
- Each of the capabilities must be implemented by modifying the client model which in return uses the observable to notify the presenters (controllers), which update the views.

Chat and Chat History:

- Fully implement the Chat feature.
- Players should be able to send chat messages at any time (even when it is not their turn). Chat messages should appear in the same order on each player's screen.

Constraints

The design documentation must be done using both UML class diagrams and sequence diagrams. It must also have **n** classes that have been commented with Javadoc for each of the **n** members of your group (part of the Phase 2 Report).

A game may have as few as 2 and as many as 5 players.

Deliverables

Design documentation

A partially running Ticket to Ride game that can be setup and whose design follows the submitted design. It must provide all of the capabilities described above.

You must provide a routine that can be run to test and demonstrate all of the client model updates and subsequent view updates (through the observable) for all capabilities described above. For every test case the routine should describe what the test case is to do, and do it. It is to then wait a sufficient amount of time (about 5 seconds) for the TA to read the description and observe the result in the GUI.