Project Overview

`GameState` object

- Array of `Player`
  - id
  - name
  - ip
  - color (hex value)
- Array of `Box`
  - owner's id
  - status
    - free, reserved, filled
  - color (set when reserved or filled)

Most logic happen in `Main.java`. The UI logic for a box is in `BoxView.java`.

There are 2 ways to update the `BoxView`. By a mouse action or when the Game State updates. Game State will override UI done by a mouse action if there's a conflict.

Server

When the user clicks on the server button `Main.onServerClicked`

- Switch to Lobby scene
- Add a player object (which contains host info) to host's game state
- Start listening for clients
  - Create 3 threads of class `ServerListener`
    - What `ServerListener` does:
      - create `output` and `input` object stream
      - indefinitely listen for incoming objects via `input`
      - once there's an incoming message (which is one of the subclasses of `Message`)
        - call `Main.onMessageReceivedFromClient`

What `Main.onMessageReceivedFromClient` does

- Check the type of incoming message
- If the message type is related to updating state of a box
  - This includes `REQUEST_BOX_RESET`, `REQUEST_BOX_FILLED`, `REQUEST_BOX_RESERVED`
  - The host updates its own Game State according to the message

    - The host then update its UI and broadcast the state to clients
- If the message type is `REQUEST_CONNECTION`
    - Which is received when a client connects to host for the first time
    - The info in this message includes the client's **name** and **IP**
    - The host then
        - adds `writer` which is an `OutputObjectStream` for the client. This allows host to send message to the client in `Main`
        - picks the **color** and **ID** for the client
        - creates `Player` object based on info above and appends the `Player` object to host's `GameState.players`
        - sends the `Player` object back to the client via `CONFIRM_CONNECTION`
        - broadcasts its own Game State to clients
- If the message type is `RECONNECT`
    - Which is received a client wants to reconnect with a new host (after the old host has disconnected). The host that receives this message is the new host that was previously a client.
    - The host broadcasts Game State to clients because `GameState.players` has changed (number of players reduced by 1)
    - The host counts number of clients it expects `expectClientCount`
    - If all clients have connected, the host broadcasts `RESUME_GAME` message to clients

Client

When the user clicks on the client button `Main.onClientClicked`

- Switch to Lobby scene
- Start listening to the host via `ClientListener`
    - What `ClientListener` does
        - create `output` and `intput` object streams
        - call `Main.setClientWriter(output)`
            - this allows client to use `output` and send message to host in `Main`
        - send `REQUEST_CONNECTION` message (contains **name** and **ip**) to the host
        - indefinitely listen for incoming objects via `input`
        - once there's an incoming message (which is one of the subclasses of `Message`)
            - call `Main.onMessageReceivedFromServer`

What `Main.onMessageReceivedFromServer` does

- Check the type of incoming message

- If the message type is `CONFIRM_CONNECTION`
    - Which is received after the client sends `REQUEST_CONNECTION` to the host
    - This message contains `Player` object for the current client
    - The client then saves this object to `Main.currentPlayer`
- If the message type is `START_GAME`
    - Received when the host clicks the start game button
    - The client will just switch to Game scene
- If the message type is `RESUME_GAME`
    - Received when the new host sends `RESUME_GAME`. The new host sends this when all clients have connected.
    - The client switches to Game scene
- If the message type is `UPDATE_STATE`
    - Received when the host broadcasts message to every client
    - The client only updates its UI

When `Main.onServerDisconnected` does

- This method is called when the host has disconnected
- The client
    - Switch to Reconnecting scene
    - If the client is the second player in `GameState.players`
        - It becomes the new host
        - and calls `listenForClients`
    - Otherwise
        - It creates a new `ClientListener` thread that connects with the new host (a 1 second delay is added before it attempts to connect to the new host). And in `ClientListender`, the client will send `RECONNECT` message to the new host

BoxView (UI for a box)

When the user clicks on the box `BoxView.onMousePressed`

- Draw a dot on the box
- call `Main.onBoxReserved`

When the user drags on the box `BoxView.onMousePressed`

- Draw a line between the current and previous points if the current point is within the box area

When the user releases the mouse `BoxView.onMouseReleased`

- If the colored area in the box is greater than the minimum percentage
    - Fill the box with current player's color
    - Call `Main.onBoxFilled`
- Otherwise
    - Fill the box with white color
    - Call `Main.onBoxReset`

What `Main.onBoxReset`, `Main.onBoxFilled`, and `Main.onBoxReserved` do

- These 3 methods get called by the UI
- If the current player is the host
    - It will update its Game State according to the method type e.g. for `Main.onBoxReset`, it will reset the box in the Game State
    - Then it will broadcast state to clients
- If the current player is a client
    - It will just send a `REQUEST_BOX_XXXX` to the host. The host will then update its own state and broadcast the updated state to all clients.

After a client receives a new state from host

- via `UPDATE_STATE` message
- if the current scene is lobby scene
    - the only change to the state that can happen when players are in lobby scene is `GameState.players`
    - when the client receives this message, it will update the UI of the lobby scene (updating list of players)
- if the current scene is game scene
    - The `GameScene` will loop through all boxes in `GameState.boxes`
        - if the box is `FREE`, then the client will fill the corresponding `BoxView` with white color (via `BoxView.reset`)
        - if the box is `RESERVED` and the current player doesn't own, it will put an X on the box (via `BoxView.reserve`)
        - if the box is `FILLED` and the current player doesn't own it, it will call `BoxView.forceFill`.
            - `BoxView.forceFill`
                - fill the box with the color of the box's owner.
                - AND if the current player is drawing in this box, their stroke will be cancelled (this forces the current player to stop drawing in the box).
                    - This is done by setting `BoxView.drawing = false`,

`BoxView.previousX = -1`, and `BoxView.previousY = -1`