Refactoring & Design Pattern List

Refactoring:

- **Single game -> multiple games**
    - Our initial code only allowed one game to exist at a time as the system was running only on localhost. With the transition to multiplayer we refactored the code to be able to work with multiple games and changed the server to handle this. This follows the observer pattern.

- **Server sends messages directly to a RollerballPanel instance -> Server sends messages to MenuGUI which sends messages to all open RollerballPanels (allowing multiple games to be open)**
    - With the refactoring of this code, the MenuGUI becomes the class that instantiates RollerballPanels. A MenuGUI keeps track of all of the open RollerballPanels and updates them as needed when it gets messages from the server.

- **Human Users -> AI player**
    - Creation of AI player class allows one AI player to play against multiple human players. Initially each human player would have its own individual instantiation of the AI class but this refactoring prevents redundancy by the AI client keeping a list of all active games

- **Move winCondition method**
    - Win condition method was moved from the client class to the game class. This follows the Information Expert/High cohesion design pattern to delegate the method to the appropriate Game class as a callable public method for that specific game instance. This avoids the redundancy of having to send a gameID from the game to the client/server to calculate if a win condition has been reached

- **Migrating code from single system to distributed**
    - Our first approach was to run a single game on the localhost and have a two clients playing the game. When transitioning to having the server host multiple games for many clients, we had to refactor the whole game creation process. Now we create games dynamically when invitations are accepted.

- **Sending Data from Client to Server**
    - Our first approach to transferring data between the client and server was to write and read from file output streams that would keep the files stored on the local device. Once we transitioned to a distributed system of clients, we couldn't store and access the same files. We changed the method of sending data from reading

and writing to files, to encoding the data to Base64 and sending those strings over tcp to their destination. The destination then decodes the information and rebuilds the objects that were transmitted.

Design Patterns:

- **Model View Presenter**
    - Our code follows a hybrid approach to the Model View Controller design pattern.
- **Observer**
    - The GUI is using listeners to run.