Abbiamo deciso di implementare sia il server RMI che il socket.
Per fare questo abbiamo due file, AppServerRMI e AppServerSocket che vengono lanciati in parallelo su due thread diversi. Abbiamo fatto in modo che i due sistemi funzionassero in maniera simultanea, implementando gli stessi metodi usando la stessa interfaccia per entrambi i sistemi. Il server socket poi ha un sistema multi-thread per gestire più connessioni in simultanea.
Una volta ricevuta una connessione, questa viene tenuta temporaneamente dalla Lobby che si aspetta di ricevere il nickname e il numero di player di quella partita. La Lobby è un singleton e non ha alcuna conoscenza di se il client sia socket o RMI, per rendere più estendibile il codice. La Lobby gestisce il riconoscimento di a che game il client sta partecipando e se un dato client sta partecipando ad un game. Inoltre tiene conto di se un client è connesso o disconnesso, per poterlo riportare agli altri giocatori e per in futuro poterlo reinserire nel game corretto in caso il suddetto client si riconnetta (o in caso di crash del server). La Lobby poi espone il controller ai vari server, facendo da “collante” tra la view e il controller, finendo il modello MVC.
Il client è implementato in maniera simile, avendo un’interfaccia che poi viene implementata dalle due tecnologie, socket o RMI, rendendo irriconoscibile alla view la differenza. Il client socket contiene un ServerStub che manda gli eventi tramite il socket, usando un oggetto SocketMessage composto da il tipo dell’evento e il payload vero e proprio. Questi poi viene riconosciuto sul server e viene chiamato il metodo corretto.