Peer review sul protocollo di comunicazione

Aspetti positivi

In generale il protocollo di comunicazione utilizzato ci sembra valido e ben strutturato.

Ci piace molto l'idea di dare all'utente la possibilità di scegliere se unirsi ad una partita già creata da qualcuno in precedenza o di crearne direttamente una nuova.

Nel nostro caso abbiamo scelto di implementare un accesso casuale alla partita, però ci sembra un valore aggiunto al gioco avere questa funzionalità in più, tant'è che stiamo pensando di implementare anche noi questa funzionalità. Dal canto nostro, possiamo consigliarvi di inserire anche un accesso casuale alle partite senza specificare la partita nella quale si vuole entrare, di modo di lasciare più libertà di scelta all'utente.

Un altro aspetto positivo che abbiamo riscontrato è relativo ai messaggi: ci sembra una scelta valida quella di utilizzare solo classi che ereditano da Message per lo scambio di informazioni. In questo modo avviene lo scambio di entità formattate e si evita di scambiare stringhe o interi in ogni fase del gioco.

Aspetti negativi

Abbiamo notato che nelle azioni di spostamento di uno Studente dall'ingresso al tavolo/isola viene inviato come parametro del messaggio una stringa contente il colore dello studente. Pensiamo sia meglio passare come parametro un intero che indichi l'indice dello studente selezionato. Questo permetterebbe, lato view, di dare la possibilità di selezionare uno specifico studente sull'ingresso, il che sarebbe, a parer nostro, di più facile implementazione in ottica GUI e ne migliorerebbe il funzionamento, risultando questa un'interfaccia più intuitiva per l'utente.

Non ci è chiaro come avviene la comunicazione dal model alla view. Più precisamente, non capiamo se è il model ad inviare i messaggi di SuccessMessage e ErroreMessage alla view, oppure se è il controller a farlo. Nel caso in cui l'implementazione da voi scelta sia la seconda, ricordiamo che nel pattern MVC è il model a notificare i cambiamenti alla view. Inoltre, non è chiaro se all'interno di questi messaggi venga inviato anche il modello, il quale è necessario alla view per mostrare lo stato aggiornato della partita. Riteniamo necessario che il modello, o sue parti, venga trasmesso al client.

Non sappiamo bene in che modo viene fatta la sincronizzazione nella ClientSocketHandler e nel server, a nostro avviso una implementazione che utilizza il costrutto synchronized come visto all'esercitazione sembra di più facile utilizzo e pensiamo possa dare una maggiore robustezza.

Infine, per quanto riguarda la gestione dei messaggi relativi ai Character, seguendo quanto da voi esposto nel modello, ci sembra che necessiti di molta logica lato client, il quale deve riconoscere il tipo di Character e strutturare le richieste di input, tuttavia ci rendiamo conto che, data l'implementazione Character, la vostra scelta di gestione dei Message risulti comoda. Tuttavia, ci sentiamo di consigliarvi di spostare questa logica di controllo lato model così da semplificare il client.

Aspetti in comune

Dall’analisi del vostro protocollo ci siamo resi conto che abbiamo in comune l’utilizzo dei messaggi serializzabili anziché l’utilizzo di stringhe formattate.

L’implementazione lato server è molto simile alla nostra, la cui idea di base è avere un server che gestisce la lobby, insieme a degli Handler che gestiscono i client. Inoltre, entrambi forniamo come funzionalità il multipartita.