PROVA FINALE DI INGEGNERIA DEL SOFTWARE PROGETTO “INVASION”

A cura di Antonio Iannacci e Marco

Il nostro progetto è articolato in queste due macrofasi principali:

1. Avvio del Server, con indirizzo preimpostato a localhost;
2. Avvio del Client, scomponibile in:
   1. Scelta della tipologia di comunicazione;
   2. Login (con eventuale registrazione);
   3. Inserimento in una lista di prepartita (con eventuale possibilità di annullare la scelta)
   4. Partita (se e solo se prepartita ha un valore valido per avviare la partita)
      1. Scelta del colore;
      2. Posizionamento delle armate;
      3. Mosse varie.

* AVVIO DEL SERVER  
  L’avvio del server è effettuato eseguendo una classe esclusivamente d’avvio che al suo interno va a creare le istanze vere e proprie del server, interfacciandosi sia col collegamento RMI (Remote Method Invocation) che col collegamento Socket: a questo punto si è pronti ad accettare le connessioni che arriveranno da più client, sia essi scelgano un collegamento RMI che Socket. In questa fase viene anche effettuata una connessione ad un Database che conterrà informazioni relative agli utenti registrati, agli utenti on-line in una sessione e alle partite che vengono effettuate.
* AVVIO DEL CLIENT  
  L’avvio del client mostra una finestra in cui è possibile selezionare il tipo di comunicazione: una volta effettuata questa scelta discriminatoria, ci si porta nella fase di login.  
  In questa fase, previo inserimento di username e password, viene effettuato il login:  
  🡪 *SOCKET*: viene inviata sul canale una stringa del tipo login@[username];[password]. Dato che non è possibili vietare alcun carattere, si è notato che i caratteri @ e ; generano problemi: la soluzione è stata quella di considerare solo la prima occorrenza del carattere @ e sostituire il carattere ; con la sequenza #;. Così facendo, però, bisogna controllare che il carattere # non sia già presente all’interno della sequenza, in particolare si trova in ultima posizione nella stringa relativa all’username: si è deciso di effettuare una pre-analisi dello username in cui si sostituisce ogni occorrenza di # con §# in modo tale da non compromettere l’integrità del ; separatore tra lo username e la password.  
  Il server, una volta ricevuta la stringa identificativa del comando, effettua l’elaborazione opposta. A questo punto è pronto per effettuare delle query al Database: viene effettuata una prima query in cui si verifica se l’utente è realmente registrato presso la piattaforma a cui segue una verifica in cui si controlla che non sia già connesso (indipendentemente dal metodo di connessione). Vengono ritornate quattro possibili risposte:  
  - ok@ : il login è stato effettuato correttamente;  
  - ko@prevLog : l’utente è registrato ma ha già effettuato il login;  
  - ko@sqlExcp : non è stato possibili effettuare il login in quanto si è verificato un problema con il Database;  
  - ko@invalidData: non è stato possibile effettuare il login in quanto uno od entrambi i dati immessi non sono corretti;  
  🡪 *RMI*: vengono effettuate delle chiamate remote che, appoggiandosi ad una classe di computazione (la stessa sfruttata tramite la comunicazione Socket), restitusce due possibili risposte tramite ritorno di un metodo:  
  - ok@ : il login è stato effettuato correttamente;  
  - ko@invalidData: il login non è stato effettuato in quanto l’utente può aver già effettuato il login oppure può aver inserito dati errati oppure ci sono stati errori imprevisti riguardanti la comunicazione con il Database.  
  Tutte queste operazioni vengono effettuate sul server, liberando il client da qualsiasi logica di gioco.  
  In alternativa al login si può effettuare la registrazione: essa prima verifica che non ci sia già un utente con lo stesso username già registrato, se questo controllo ha esito negativo si prendono lo username e la password e si registrano sul Database.   
  Una volta effettuato il login, si passa ad una finestra di attesa: si è deciso di inserire dapprima l’utente in una lista d’attesa. Ogni volta che si inserisce l’utente in questa lista, si verifica che il resto della dimensione della lista diviso il numero di utenti per quella partita sia pari a 0: in caso affermativo, si lanciano tanti thread contenenti nuove partite quanto è il valore della dimensione della lista degli utenti in attesa diviso il numero di utenti. Prima di fare ciò, però, gli utenti vengono rimossi dalla lista d’attesa e messi in una lista interna alla partita che verrà usata come ordine di gioco. Se si pensa di aver trascorso troppo tempo in attesa, è possibile abbandonare la lista d’attesa relativa a quella partita ed inserirsi in una nuova coda.  
  Una volta che la partita è stata avviata, si seleziona il colore tra 6 possibili valori (con i valori opportunamente disabilitati in base alle scelte effettuate dagli altri utenti). Una volta che tutti hanno effettuato questa scelta, viene mostrata la mappa.  
  La mappa è dotata di 42 JTextField, opportunamente colorati con il colore scelto in precedenza ed opportunamente editabili, dato che sono in mio possesso. Una volta mostrato un messaggio che mi informa che sono nel mio turno, posso iniziare a posizionare le armate: quando penso di averne inserite a sufficienza, premo un pulsante di salvataggio che recupera i valori da me inseriti e controlla che tutti abbiano un valore maggiore od uguale ad una armata e che la somma totale delle pedine non superi quella consentita. Nel caso in cui una di queste condizioni viene meno, si effettuerà un posizionamento automatico. Il posizionamento automatico dovrebbe essere svolto anche quando scade un timer di 60sec da avviare ad inizio del posizionamento che, però, non abbiamo gestito poiché la soluzione da noi adottata (ovvero avviare un thread controllore del timer dalla Mappa) rallentava di molto l’applicazione, rendendola ingiocabile. Troppo tardi abbiamo pensato ad una soluzione alternativa che non siamo riusciti ad implementare. È comunque presente l’implementazione del timer originario. Una volta che tutti gli utenti affiliati ad una stessa partita hanno completato il posizionamento, viene inviato un comando di aggiornamento che mostra, sulla mappa, le scelte degli avversari.  
  A questo punto inizia la fare di gioco vera e propria (priva del timer controllore di 30sec poiché anch’esso rendeva il tutto ingiocabile): è possibili effettuare tre azioni.  
  🡪 *ATTACCO*: premuto il pulsante relativo, tramite un menù a tendina si seleziona il territorio da cui effettuare l’attacco. Si invia un comando al Server (tramite il canale di comunicazione oppure tramite una callback) pari a ‘attacca@territorioSelezionato’. Il Server elabora questo comando e risponde con ‘attaccabili@[t1, t2, …]’, ovvero la lista di tutti i territori che posso attaccare. Nel caso in cui questa lista sia vuota, si consente all’utente di selezionare un nuovo territorio e rieseguire la verifica. Se, invece, la lista contiene valori validi questi vengono inseriti in un secondo menù a tendina da cui si seleziona il territorio che voglio attaccare. A questo punto abilito una casella di testo in cui inserisco il numero di pedine e posso premere il pulsante per l’invio del comando ‘attacco@territorioAttaccante;territorioDifensore;numeroPedine’. Il numero di pedine è opportunamente settato in modo che possa rispettare le consegne della specifica.  
  Il Server elabora questo comando ricercando l’utente a cui appartiene il territorioDifensore: esso viene contattato con con il comando ‘difesa@territorioAttaccante;territorioDifensore;numeroPedine’ a cui l’utente risponderà, tramite una finestra di conferma, il numero di pedine con cui vorrà difendersi (opportunamente controllate in base al numero di pedine presenti sul territorio).  
  A questo punto si lanciano i dadi e si comunica ad entrambi gli utenti coinvolti nella mossa, la sequenza ‘dado@attacco:d1;d2;d3#difesa:d1;d2;d3’: essi la elaborano e settano dei campi opportuni sulla grafica. Successivamente il server studia il lancio dei dadi ed invia un comando per identificare il vincitore: gli utenti mostrano la elaborano e mostrano a video una finestra in cui si visualizzano le schermate perse da ciascuno. Alla fine del di questa mossa, viene aggiornata la grafica.  
  🡪 *SPOSTA*: come prima, solo che i comandi sono ‘sposta’ e ‘spostabili’ e non vi e la parte relativa ai dadi. Per capire se è possibile effettuare tale mossa, abbiamo ideato un algoritmo di visita in profondità ricorsivo sui territori appartenenti ad un dato utente. Alla fine della mossa, il turno viene passato al giocatore successivo. Alla fine di questa mossa, viene aggiornata la grafica di ogni utente.  
  🡪 *FINE*: passa il turno al giocatore successivo tramite un semplice comando ‘fine@’. Alla fine della mossa, il turno viene passato al giocatore successivo. Alla fine di questa mossa, viene aggiornata la grafica di ogni utente.  
  🡪 *RICARICA*: deciso un bonus di 5 armate, si vede se l’utente possiede tutti i territori di un continente. In caso affermativo, si somma questo bonus a quello previsto nelle specifiche. In caso negativo si assegna solo quello previsto nelle specifiche. L’utente inserisce n armate sui suoi territori e preme l’apposito bottone di Ricarica, il client prende i valori inseriti e verifica che non siano state inserite più armate del dovuto. In caso ciò sia stato fatto, l’utente viene punito sottraendogli le truppe bonus. Segue un aggiornamento della grafica per tutti gli utenti e si ha la nuova possibilità di effettuare le mosse appena descritte.  
  Un utente viene distrutto quando la lista dei suoi territori ha dimensione nulla, segue una notifica all’utente stesso e una rimozione dalla lista dei giocatori della partita. La partita termina quando nella lista dei giocatori vi è un solo utente. Per gli ultimi due utenti sopravvissuti, vengono inseriti in una entry sul Database nella tabella partita in cui si.  
    
  ATTENZIONE: è necessario, nel momento del comando RICARICA, attendere che compaia la finestra che indica le armate bonus a disposizione ed il pulsante Ricarica si attivi.  
  Questo è stata la nostra idea di implementazione, siamo consapevoli di aver tralasciato alcuni aspetti (ad esempio i timer o la gestione delle eccezioni nel caso di chiusure impreviste del server o di particolari finestre) ma, speriamo, che il nostro lavoro venga comunque apprezzato.

Antonio Iannacci – Marco Rodella