Software Design Document di Dinosaur Island

by Riccardo Ancona and Alessandro Ditta

.: Server Side

- *Avvio del server:* una volta lanciata l'applicazione, la sua finestra principale permette di modificare il numero massimo di connessioni gestitibili dal server, le porte di comunicazione socket e RMI e l'indirizzo al quale effettuare il bind dello stub RMI del server e l' RMI registry . Il pulsante start permette l'avvio del server: durante l'avvio vengono caricati i dati della partita salvati durante l'ultimo shutdown effettuato correttamente (i file sono `scores.dat`, `players.dat` e `map.dat`). Se non sono presenti dati salvati in precedenza, viene generata una nuova mappa casuale e il server si mette in attesa di nuove connessioni, sia socket che RMI . Se solo uno tra `players.dat` e `map.dat` dovesse mancare all'avvio del server, la procedura non può proseguire: per permettere il ripristino della funzionalità di avvio occorre eliminare anche l'altro file presente, quindi far ripartire il server.
Nota bene: non è necessario far partire il comando rmiregistry da terminale in quanto il programma crea ed esporta un nuovo rmi registry direttamente via codice nel metodo startUp() della classe `RMIManager`. Le impostazioni di sicurezza del registro RMI vengono anch'esse assegnate via codice (non sono quindi passate come argomenti alla jvm).
- *Gestione delle connessioni:* le classi `SocketManager` e `RMIManager` si occupano di accettare le connessioni in arrivo e di assegnarle ad un thread autonomo (la classe `ClientHandler`) che gestisca la connessione con il client. Le sottoclassi `SocketClientHandler` e `RMIClientHandler` gestiscono la comunicazione in maniera appropriata, costituendo di fatto un Wrapper per i comandi del `ClientHandler`, che prescindono dal tipo di connessione utilizzata. Ogni azione compiuta dal server viene registrata in un apposito log che viene salvato su disco e che è possibile visualizzare a video nella schermata principale. Ogni `SocketClientHandler`, dopo la creazione, attende la ricezione di una richiesta dal client: a ricezione avvenuta, viene effettuato il parsing della richiesta dalla classe `ServerRequestParser`, che analizza la prima parte della richiesta e controlla che esista un oggetto , di tipo `RequestFactory`, in grado di creare un oggetto `Request` che espleti tale richiesta. Se tale oggetto esiste viene chiamato il metodo `validate()` sul messaggio ricevuto dal client il quale verifica la correttezza sintattica del messaggio. Se la verifica ha esito positivo, viene creato un oggetto di tipo `Request` corrispondente alla richiesta del client, quindi viene eseguito il metodo `execute()` di tale oggetto che espleta le istruzioni previste dalla richiesta del client. Al termine delle istruzioni viene generato un oggetto `Response` contenente il messaggio di risposta del server, che viene quindi inviato al client.
Nota bene: nei comandi nei quali è previsto l'inserimento di username, password, token, ID dinosauri, la specifica prevedeva che tali campi fossero alfanumerici, tuttavia non spiegava il significato esatto di tale termine: pertanto è stata adottata la convenzione di considerare alfanumerica una qualsiasi stringa che contenga solamente caratteri dell'alfabeto inglese minuscoli o maiuscoli oppure cifre decimali da 0 a 9, in qualsiasi ordine. Per effettuare un controllo di validità su tali campi, il confronto viene effettuato con l'espressione regolare `[a-zA-Z0-9_]+`. Stringhe contenenti caratteri non validi genereranno un errore di sintassi al momento della validazione, che porterà il server ad una risposta di tipo @comandoNonValido .
- *Logica di gioco:* la logica di gioco segue fedelmente la versione 1.5 delle specifiche fornite, tuttavia alcune parti della specifica non definivano in dettaglio certi comportamenti della logica di gioco. Di seguito viene riportato un elenco di comportamenti adottati dalla logica del server riguardanti parti poco chiare della specifica.

Ambito	Comportamento discutibile	Implementazione effettiva
Logout in partita	Non è descritto cosa succede se un giocatore in partita richiede il logout senza richiedere prima il comando @uscitaPartita	Il giocatore in partita che richiede un logout viene prima rimosso dalla partita quindi disconnesso dal server.
Età del dinosauro	L'età iniziale del dinosauro al momento della sua creazione	Ogni dinosauro viene creato con età pari a 0.
Deposizione uovo	Il comando @deponiUovo restituisce l'ID del nuovo dinosauro, il che fa supporre che il nuovo dinosauro sia stato creato, e che dunque possa venire effettuato il comando @statoDinosauro con il nuovo ID, tuttavia nelle pagine precedenti è scritto che il nuovo dinosauro viene creato al turno successivo: dunque un comando @statoDinosauro eseguito subito dopo il comando @deponiUovo non può restituire ID non valido, poiché l'ID è stato appena restituito da una risposta del server, ma non può nemmeno fornire	Il dinosauro creato dopo il comando @deponiUovo viene creato e posizionato immediatamente sulla mappa, tuttavia non può eseguire alcuna azione nel turno corrente, dunque è effettivamente disponibile nel turno successivo a quello della sua creazione.

	informazioni riguardo la posizione del neonato dinosauro.	
Movimento dinosauro	Le specifiche prevededono che un dinosauro possa muoversi nel quadrato di lato 5 o 7 caselle che lo circonda, tuttavia non specificano chiaramente se un dinosauro possa muoversi sul suo stesso posto oppure no.	Il dinosauro non può muoversi sul suo stesso posto: la sua posizione attuale costituisce infatti una posizione non valida per il movimento.

.: Communication (Parti condivise tra Server and Client)

In questo progetto sono contenute le classi condivise tra client e server, in particolare sono state condivise le risposte che il server invia al client (e che dunque il client deve analizzare anch'esso) nel package `commands`, i beans che vengono scambiati tra client e server (che contengono alcuni contenuti delle risposte del server) nel package `beans`, le eccezioni generate dalla logica di gioco nel package `exceptions`, le interfacce del client e del server per la comunicazione RMI nel package `communication`, le interfacce che riguardano il logging delle operazioni eseguite nel package `logging`, e infine alcune classi utili sia al client che al server nel package `utils`.

.: Client Side

- *Avvio del client:* una volta lanciata l'applicazione, il client carica, se presente, un file contenenti i settaggi precedentemente salvati delle opzioni di connessione e dell'ultima coppia username e password correttamente salvata. Cliccando sull'apposito tab, è possibile scegliere di modificare le impostazioni di connessione al server, quali l'indirizzo IP, la porta di comunicazione, e il metodo di connessione (socket o RMI). Una volta definite le opzioni di connessione, è possibile registrarsi al server oppure effettuare il login. Una volta effettuato il login, viene aperta una finestra di attesa nel quale il giocatore può vedere la classifica e i giocatori in partita, può decidere di effettuare un logout oppure può decidere di entrare in partita. Se l'utente cerca di entrare in partita senza aver creato prima una specie, viene aperta una form che permette la creazione di una nuova specie. Una volta entrato in partita, viene aperta una finestra in fullscreen 800x600x32bit che consente una maggiore coinvolgimento del videogiocatore.
- *Gestione della connessione:* la struttura delle classi che gestiscono la connessione con il server è speculare a quella del server. Le classi `SocketServerHandler` e `RMIServerHandler` si occupano, in mutua esclusione, di gestire la connessione con il server. Per quanto riguarda la tipologia di connessione Socket, è previsto che durante la registrazione di un utente si instauri una connessione socket che permetta il solo scambio delle informazioni di registrazione (quindi dopo aver ricevuto una risposta dal server riguardante la registrazione il socket viene chiuso), mentre è prevista una instaurazione di connessione continuativa soltanto dopo aver effettuato il login, questo per dare all'utente l'eventuale possibilità di poter modificare le impostazioni di connessione dopo aver provveduto alla registrazione di un nuovo utente.
Ogni comando della classe `SocketServerHandler` che invia una richiesta al server invoca un metodo `sendRequestAndWaitResponse()` che prende come argomenti il messaggio da inviare e un oggetto di tipo `ResponseFactory` in grado di capire se la risposta che il server invierà in risposta è valida per la richiesta effettuata oppure no. Una volta inviata la richiesta il thread che l'ha inviata si mette in attesa tramite un' istruzione `wait()` che arrivi qualcosa sul client. Un apposito thread a parte è responsabile di ricevere tutti i messaggi del server in entrata e di controllarne la tipologia, discriminando se tale messaggio è un cambio turno oppure no. Se non si tratta di un cambio turno, esso risveglia tutti i thread che si erano messi in attesa tramite l'istruzione `notifyAll()` e condividerà con loro il messaggio ricevuto: dunque ogni thread risvegliato controllerà che il messaggio ricevuto possa corrispondere ad una risposta valida, e in caso negativo ritornerà in `wait()` aspettando un nuovo messaggio dal server; in caso positivo analizzerà la risposta ricevuta e la fornirà all'interfaccia grafica che l'aveva richiesta. Se invece il messaggio ricevuto dal server è un cambio turno, non viene risvegliato alcun thread e il `SocketServerHandler` setta la sua variabile `changingTurn` a true. Il gestore della sessione di gioco prrovederà a leggere il valore della variabile changingTurn e a comportarsi di conseguenza gestendo o meno il cambio turno.
- *Sessione di gioco:* Durante la sessione di gioco, gli unici comandi da tastiera disponibili sono quelli indicati dalle informazioni in sovraimpressione sul video, a parte il tasto ESC che può essere premuto per tornare alla sala di attesa. I comandi impartibili via mouse invece sono lo scorrimento della mappa tramite la pressione prolungata del tasto destro del mouse sull'area di gioco, la selezione/deselezione di un dinosauro tramite la pressione del tasto sinistro del mouse sull'icona rappresentante un dinosauro. Quando si seleziona un dinosauro, vengono mostrare delle informazioni al riguardo, insieme a tre icone rappresentanti le azioni che tale dinosauro può eseguire: se una azione è grigia, vuol dire che non è eseguibile. Se si seleziona l'azione di movimento di un dinosauro, per ultimare l'operazione di movimento è necessario cliccare con il tasto sinistro sulla cella di destinazione.

- *Architettura ingame:* la sessione di gioco viene fatta partire su un thread separato, che richiama il metodo `run()` della classe `GameManager`. Tale metodo provvede a far partire il motore di gioco (ereditato dalla classe `GameCore`) chiamando il metodo `init()`, che inizializza alcune classi, tra cui `ScreenManager` che si occupa di riservare una finestra fullscreen al gioco, `InputManager` che si occupa della gestione degli input da tastiera e da mouse, `ResourceManager` che si occupa del caricamento delle risorse grafiche dal disco fisso nella memoria video, `TileMap` che contiene le informazioni sulla mappa del server, e `GameRenderer` che si occupa del disegno degli elementi di gioco. Al termine del metodo `init()` viene innescato il game loop, nel quale, step per step, viene aggiornato lo stato del gioco tramite la funzione `update()` che chiama a sua volta chiama il metodo `checkInput()` che controlla gli input forniti dal giocatore, e quindi al termine della funzione update viene invocato il metodo `draw()` che ridisegna lo schermo. Per evitare l'effetto di sfarfallio, lo schermo viene ridisegnato utilizzando una tecnica di double buffering.

.: Note sui diagrammi UML forniti

I diagrammi `Dependencies.png` contengono solo le dipendenze tra i package più importanti relativi ciascun progetto. Inoltre, per ogni progetto sono stati creati solo quei class diagram più significativi, cercando di bilanciare completezza e leggibilità, tuttavia a causa di continui malfunzionamenti del plugin eUML2, non è garantita la perfetta corrispondenza 1 a 1 tra codice e diagramma. Infine è stato creato un sequence diagram che illustra una possibile sessione di gioco completa tra l'utente e il gioco.