JUSTPOKER™

Progetto ingegneria del software

Componenti progetto

Alessandro Cardillo Ottaviano: 1055164
Dolete Marian Stefan: 1093398
Leonardo Forchini: 1092237
Oleh Kalkovets Cestonaro: 1092174

Obiettivo

L’obiettivo di questo progetto è quello di poter fornire a un casinò un nuovo modo per poter diffondere il gioco del Poker all’italiana dando la possibilità ai propri clienti di utilizzare il servizio in modo gratuito e senza rischi di perdita economica tramite un sistema client/server stabile per il gioco online multigiocatore.

Difficoltà incontrate

Le principali difficoltà che abbiamo incontrato sono state:

- La creazione e gestione del server in cloud
- La programmazione del codice secondo l'architettura client/server e non con il solito main procedurale
- Familiarizzare con la GUI di JavaFX

Paradigma di programmazione e modellazione utilizzati e tools

La richiesta del progetto era quello di utilizzare il paradigma orientato agli oggetti con il linguaggio Java, mentre la modellazione è stata gestita tramite diagrammi UML.

Tool principali:

- IDE: Eclipse
- Modellazione: PlantUML, Papyrus, LucidChart
- Manutenzione: STAN, PMD
- Testing: JUnit
- Server: Oracle in cloud
- Dipendenze varie: H2, Log4j, JavaFX

Software configuration management

La richiesta del progetto era quello di utilizzare di utilizzare GitHUB come software per la gestione della configurazione. Abbiamo utilizzato molte delle sue funzionalità:

- clone: per avere una copia della repo remota sul proprio pc
- commit/push: per inviare le proprie modifiche alla repo remota
- pull: per aggiornare la copia locale della repo remota
- branch: ci ha permesso di creare diversi branch a partire dal main per lo sviluppo parallelo di diversi componenti del sistema
- merge: per unire un branch di sviluppo al main
- Kanban board: per tenere monitorato lo stato e il carico di lavoro durante le Sprint

Software life cycle

Abbiamo scelto di utilizzare il metodo di processo Agile SCRUM.
Il lavoro è stato diviso in 7 Sprint:

- le prime si concentrano principalmente sulla documentazione
- le successive sulla modellazione UML e conseguentemente su scrittura/testing/manutenzione del codice, con aggiornamento della manutenzione

Questo approccio ci ha permesso di ridurre i rischi di errore e di migliorare la coordinazione del team

Requisiti

Per l'elicitazione dei requisiti ci siamo basati sull'analisi delle regole del gioco, facendo brainstorming e descrivendo il flusso di gioco in linguaggio naturale.
I requisiti sono stati poi inseriti nel relativo documento dei requisiti utilizzando lo standard IEEE 830, che ne definisce la struttura.

Architettura

Lo stile Architetturale utilizzato è Client/Server basato su Socket:

- il Server è ciò che gestisce la logica del gioco, il flusso e l'accesso al database (è il componente che definisce la verità assoluta sullo stato del gioco)
- il Client si occupa dell'invio dei comandi al server
- la GUI risponde agli stimoli del Client è mostra lo stato del gioco

La comunicazione avviene tramite scambio di messaggi (dati serializzati) sulla rete

Design pattern

Abbiamo utilizzato i seguenti pattern:

- Singleton: per l'istanza del server che è unica e che deve poter gestire la coordinazione dello stato globale
- Abstraction occurrence: per rappresentare l'astrazione di diversi elementi di gioco gestendo i suoi diversi stati durante l'evoluzione della partita
- Observer Pattern:
  - per fare in modo che la GUI rispondesse agli input del Client
  - per fare in modo che il singleton Server potesse comunicare con il proprio ServerManager per aprire nuove connessioni alla fine di una partita

Metriche di qualità

- I package Server e GUI hanno valori alti di CC ed ELOC poiché assumono un controllo centralizzato dello stato del gioco e del sistema
- Il package Server ha anche un valore alto di FAT poiché si assume tutte le responsabilità nella gestione del gioco
- I package Database e Card.Model hanno invece una D che indica la loro estrema concretezza e stabilità, dovute al loro ruolo (il primo è un componente di basso livello, mentre il secondo di modello)
- Non sono presenti dipendenze cicliche tra package

Modellazione

Nella modellazione abbiamo prodotto tutti i diagrammi UML:

- Use Case: per le azioni di gioco che il player può effettuare
- Class: per la generazione del codice con papyrus
- Final State: per gli stati della partita/giocatore durante le fasi di gioco
- Communication: per la comunicazione client/server/database
- Sequence: per la sequenza dei messaggi tra client/server/database
- Activity: per illustrare il flusso di gioco gestito dal server
- Package: mostra le relazioni tra i vari package e classi
- Component: mostra le interazioni tra i componenti del sistema

Implementazione

Per implementare i requisiti abbiamo utilizzato l'IDE Eclipse: abbiamo programmato in Java, creando un progetto Maven che ci ha aiutato a gestire al meglio le dipendenze (Junit, JavaFX, H2, Log4j).
Il server è un server cloud su una macchina virtuale Ubuntu, mentre il database è embedded.

Testing

Abbiamo scritto manualmente i casi di test JUnit, che ci hanno permesso di trovare e risolvere molti bug presenti nel codice. Principalmente sono stati testati i metodi centrali nella logica e nel flusso di gioco. Lo scambio di messaggi vero e proprio tra client e server è stato testato a posteriori una volta che la GUI è stata ultimata.

DEMO