Relazione fine Fase1-ISS25

Repository Git: https://github.com/gioliee/IngegneriaDeiSistemiSoftware25

Riassunto del lavoro eseguito:

L'applicazione ConwayLife25 simula il gioco della vita ovvero un'evoluzione cellulare all'interno di una griglia utilizzando Java come linguaggio principale.
Inizialmente si è andato a costruire un progetto Java riguardante la logica, ovvero il core business, del gioco, che, come output, aveva un System.out in console. A seguire è stato eseguito un refactoring del progetto introducendo i concetti di Cella e Griglia applicando dunque il principio di Singola responsabilità cosicché ogni file si occupasse di un aspetto specifico del sistema. Inoltre, essendo che il vero obiettivo era la costruzione di una GUI, si è aggiunta un'interfaccia per l'output così da poter utilizzare diversi dispositivi di output qualora fosse necessario, iniziando a separare la logica di rappresentazione da quella del gioco. Finita la parte di logica è stato opportuno eseguire dei test così da validare il lavoro fatto per poter poi passare alla progettazione della GUI. Per gestire il nuovo progetto, inoltre, si è dovuto eseguire un'analisi del problema così da evidenziare bene come procedere facendosi opportune domande per creare un piano d'azione ben strutturato, come ad esempio chi comanda il gioco o come gestire la comunicazione.
Per il progetto della GUI, inizialmente, si è optato per l'utilizzo del framework Springboot che ci garantisce di strutturare l'architettura in layer dedicati, ad esempio, nel nostro caso, uno dedicato alla logica di business e uno alla GUI. Infatti, nello strato di business logic abbiamo potuto inserire il progetto precedentemente effettuato. Ciò però ci portava ancora ad avere un sistema monolitico e dipendente dove frontend e backend si parlavano tramite WebSocket. Si è poi pensato di introdurre un pattern MVC dove il model fosse la logica di business, la view la pagina html e il controller un controller Spring. Arrivati a questo punto avevamo un tipo di interazione human to machine, ma per passare a un primo approccio di interazione machine to machine si è implementata una classe caller. Introducendo il concetto di microservizi si è iniziato a separare la GUI e la logica del gioco in due microservizi indipendenti. Affinché potessero comunicare in modo efficace si è utilizzato il protocollo MQTT per lo scambio di messaggi introducendo, inoltre, astrazioni di comunicazione avanzate in quanto gli è stato affiancato una libreria custom.

Le finalità di questa prima fase sono state la realizzazione bottom-up del gioco Conway introducendo via via sempre più caratteristiche che portassero complessità al problema.
Inizialmente si è partiti da come funzionasse effettivamente il gioco così da avere una base di lavoro e un primo approccio al sistema. Infatti, l'obiettivo era creare un sistema manutenibile, estendibile e modulare, affinché progressivamente si potesse aggiungere complessità al problema, il tutto passando da una fase di testing cosicché ciò che era stato testato non dovesse essere modificato. Per effettuare ciò ci si è basati anche sui principi di buona progettazione del software come il principio di Singola responsabilità. Un altro scopo di questa prima fase è stata la creazione di una pagina HTML che fungesse da input-output lato utente. Questa doveva interagire con il server inizialmente tramite WebSocket. Successivamente l'obiettivo era trasformare il sistema in un microservizio usando SpringBoot in cui le varie comunicazione fossero gestite tramite scambi di messaggi usando MQTT. Successivamente, si è passato alla costruzione di due microservizi in modo da dividere la parte della GUI con il resto del controllo del gioco, entrambi in comunicazione attraverso messaggi usando MQTT e astrazioni di comunicazione.
Nelle prossime fasi l'obiettivo sarà trasformare l'applicazione Java in attore usando il linguaggio custom qak così da essere capace di interagire con l'esterno mediante scambio di messaggi ma senza l'uso di Spring. In seguito, poi le singole celle verranno distribuite su più dispositivi fisici, ciò

ci permetterà di nascondere i dettagli implementativi del sistema, a seguito di vari refactoring, concentrandoci solo sulle interazioni di alto livello. In questo modo si andrà man mano a costruire un sistema flessibile, scalabile ed estensibile basato su attori.

Tutti i sistemi visti a lezione sono stati sperimentati durante la prima fase del corso, anche se si è avuto qualche problema nell'avvio del progetto Spring che utilizzava WebSocket per comunicare tra business logic e GUI in quanto la connessione sulla WebSocket risultava fallita. Nonostante ciò, si è riuscito ad effettuare in autonomia il refactoring del sistema ConwayLife25 introducendo Cell e Grid e togliendo in tutta la logica di business ciò che riguardava la rappresentazione del gioco più a basso livello. Per quanto riguarda la parte di MQTT essendo una tecnologia nuova si è aspettato di discutere il suo funzionamento durante la lezione per capire bene con che strumento si avesse a che fare.

Dal punto di vista tecnologico si sono sperimentate nuove tecnologie, ma si sono anche viste tecnologie già conosciute ma a un livello di complessità differente. Partendo in ordine cronologico si è preso familiarità con il sistema di build automation Gradle per la costruzione di progetti ben organizzati e strutturati e la gestione delle dipendenze. Nonostante la conoscenza di Maven e quindi avendo già concettualmente chiaro cosa facesse il sistema alla base, si è trovato Gradle leggermente più complesso inizialmente, ma più potente e flessibile. Successivamente si è utilizzato il framework Springboot per la costruzione di microservizi rendendo il progetto scalabile. Si è inoltre visto e fatto uso di Websocket già utilizzate in passato, ma non in un contesto con elevata complessità come ConwaLife25. Congiuntamente si è presa familiarità con un nuovo protocollo per lo scambio di messaggi, MQTT. Ciò ha permesso di far interagire i vari microservizi attraverso messaggi ovvero stringhe opportunatamente strutturate. Inoltre, si avevano conoscenze pregresse su Git e Docker.
Dal punto di vista concettuale si è in primis imparato a costruire progressivamente un sistema complesso concentrandosi sull'analisi del problema affinché si sviluppasse codice riutilizzabile, manutenibile e scalabile attraverso anche quelli che sono i principi dell'ingegneria del software. Si è inoltre puntato molto sullo sviluppo in parallelo di documentazione adeguata riguardante il codice e le varie scelte implementative. Altro aspetto importante è stato l'imparare a pensare in termini distribuiti introducendo il concetto di microservizio come entità a sé stante in grado però di comunicare con l'esterno.

È stato scelto come modello di studio il gioco della vita di Conway probabilmente per il suo alto grado di complessità. Risulta innanzitutto imprevedibile nella sua evoluzione visto che configurazioni simile possono portare a risultati completamente differenti, ma il gioco inoltre risulta essere estremamente variegato vista l'esponenzialità delle varie configurazioni che possono essere adottate. Inoltre, al nostro scopo risulta anche ben applicabile il principio di costruzione in progressione del software.

L'utilizzo di Java alla base può partire inizialmente dalla conoscenza che si ha di questo linguaggio in quanto largamente utilizzato nei precedenti corsi di studio, è un linguaggio a oggetti semplice che poteva ben adattarsi al nostro caso di studio. Esistono sicuramente tecnologie più moderne ed efficienti che potrebbero sviluppare il nostro gioco, ma associare a Java anche un framework come Springboot permette di mettere un focus su come organizzare una buona architettura di lavoro, scalabile e manutenibile, perdendo in interattività e velocità rispetto ad altre soluzioni implementative.

Key point sull'ingegneria del software durante la fase 1:

- Procedere in modo bottom-up nella realizzazione del progetto
- Applicare il principio di Singola Responsabilità
- Dependency injection grazie all'uso di un framework
- Applicazione del Pattern Observer
- Applicazione del Pattern MVC
- Separazione tra logica del gioco e rappresentazione del gioco
- Design for change

Si pensa che conwayguialone e conway25JavaMqtt siano due microservizi poiché:

- Sono indipendenti e deployabili separatamente e dunque aggiornati, scalati e distribuiti senza influenzare altri servizi
- Potenzialmente si potrebbe far uso di linguaggi di programmazione diversi anche se abbiamo scelto per altre strade
- Hanno compiti ben definiti e limitati (Single Responsability Principle)
- Comunicazione via API asincrone utilizzando protocolli standard come MQTT

Lo sviluppo di una libreria custom per la comunicazione ha diversi vantaggi in sistemi complessi in quanto nascondono la complessità dei protocolli di comunicazione che si decide di usare, rendendo semplici ed automatiche diverse operazioni. Questo inoltre rende il tutto riutilizzabile visto che sotto avremmo avuto diverse duplicazioni di codice. Inoltre, queste astrazioni ci permettono di rendere il sistema adatto a diversi protocolli in modo uniforme. Un esempio è la libreria unibo.basicomm23-1.0.jar che usa il concetto di interconnessione come canale bidirezionale in grado di astrarre la comunicazione garantendo l'uso, dunque, di diversi protocolli come sua implementazione.

Valutazione:

B+, in quanto la parte opzionale non è stata svolta e inoltre non ho citato l'utilizzo del pattern Singleton