Relazione Fase1-ISS25

Sebastiano Giannitti
Repository GitHub: https://github.com/sebastianogiannitti/iss

Durante la fase iniziale del progetto ConwayGUI, abbiamo analizzato e definito una prima soluzione sfruttando linguaggi noti come JavaScript e Java per sviluppare il gioco Conway Life. L'attenzione si è concentrata sui requisiti del sistema, con l'obiettivo di separare le diverse funzionalità del progetto.

Sfruttando le funzionalità di java (interpretando celle e griglia come Oggetti) introduciamo SpringBoot che integra queste funzionalità in un sistema più strutturato e rendendo possibile lo sviluppo di una pagina HTML che funga da interfaccia di input-output, interagendo con il server tramite WebSocket per offrire un'esperienza interattiva (owner) o di mirroring (non owner). Inoltre, si è avviata la trasformazione del sistema in un microservizio autonomo basato su Spring Boot, con la possibilità di comunicare tramite protocollo MQTT (meccanismo publish/subscribe con brokerMqtt).

Questo ha introdotto la necessità di distribuire il software come immagini Docker (tramite DockerFile) che vengono eseguite in container, che a loro volta comunicano grazie a Docker Compose. Il tutto facilitandone la distribuzione e l’esecuzione su diversi ambienti.

Sistemi realizzati e sperimentati

Durante la prima fase del corso, ho avuto l'opportunità di lavorare su diversi sistemi, tra cui:

Conway25Java: Ho realizzato le classi Java che formano il nucleo del gioco Life di Conway. Questo progetto è stata la mia principale realizzazione finora e mi ha permesso di approfondire le competenze nella logica di programmazione e nella gestione delle regole di gioco.

ConwayGUI: Ho cercato di sviluppare un sistema Spring Boot che integra Conway25Java, fornendo una GUI come dispositivo di input-output. Sebbene abbia seguito le lezioni e compreso i concetti, non sono riuscito a portare a termine questa parte autonomamente.

Conway25JavaMqtt: Ho iniziato a esplorare la creazione di un'applicazione standalone che comunica via MQTT con il mondo esterno. Tuttavia, non sono ancora riuscito a implementare questo sistema in modo indipendente.

ConwayGuiAlone: Ho avviato lo sviluppo di un servizio Spring Boot progettato per offrire la GUI e interagire via MQTT con Conway25JavaMqtt.

Nonostante abbia trovato difficoltà, seguire lo sviluppo di ConwayGUI mi ha permesso di imparare concetti chiave riguardo l’architettura software e la comunicazione tra componenti.

Key Points

- **Analisi dei requisiti** Il primo passo è stato identificare il core business, ovvero la logica centrale del gioco di Conway. Questo ha significato suddividere il sistema in domini separati, evitando di mescolare logica di gioco, interfaccia utente e gestione della comunicazione. L'obiettivo era garantire una progettazione modulare, dove ogni componente fosse autonomo e con responsabilità ben definite.
- **Principio delle Singole responsabilità** Per mantenere la chiarezza del sistema, è stato adottato il modello MVC. Il Model contiene la logica del gioco (Conway's Life) ed è completamente separato dalla GUI. La View (GUI), realizzata in HTML funge da interfaccia grafica per l'utente, mostrando lo stato del gioco e permettendo l'interazione, pensata come un semplice strumento di input/output. Il Controller è implementato con Spring Boot, gestisce l'interazione tra la GUI e il core business. Riceve i comandi dall'utente (es. "Start", "Stop", "Clear") e li inoltra alla logica di gioco.
- **Principio delle dipendenze** L'interfaccia tra l'applicazione e la GUI non è stata costruita attorno a una tecnologia specifica, ma attorno a un modello astratto, rendendo il sistema più adattabile. Questo significa che è stato il core business a stabilire le regole dell'interazione, e non il contrario.
- **Prototipi e verifiche** Lo sviluppo della GUI è stato affrontato in modo incrementale, partendo da prototipi per validare le funzionalità prima di realizzare la versione definitiva. La prima verifica è stata una semplice stampa su console, per testare la logica del gioco. Solo in seguito è stata introdotta l'interfaccia grafica per migliorare l'interazione utente.
- **Concetto di "armatura"** Dopo aver costruito la logica di gioco, si è passati alla fase di integrazione con la GUI. L'idea era quella di non modificare il codice del core business, ma di aggiungere una struttura esterna, una sorta di "armatura" (da qui il concetto di Ironman). Per realizzare questo, è stato scelto Spring Boot, un framework che permette di gestire facilmente la comunicazione tra GUI e core business. A questo punto, è stato fondamentale capire come Spring Boot si relazionasse con i diversi componenti dell'applicazione: il controller intercetta le azioni dell'utente (es. "stop", "clear", …) e le trasmette alla logica di gioco. Il flusso non è più basato su un ciclo, ma è controllato direttamente dagli input dell'utente, incarnando il concetto di Human-Machine Interface.
- **Interazione tramite WebSocket** Esistono due modalità per collegare l'HMI con il core business: HTTP (sincrono) con un'interazione più lenta e meno dinamica, oppure tramite WebSocket (asincrono) che permette una comunicazione bidirezionale in tempo reale, senza attese tra richiesta e risposta. Nel nostro progetto, abbiamo scelto WebSocket per migliorare la reattività e consentire una gestione più fluida dell'interazione utente-macchina. Inoltre, il controller non restituisce pagine HTML, ma un JSON con i dati del sistema, che consente l'elaborazione da parte di altre applicazioni.
- **Architettura globale** A questo punto, l'architettura del sistema è più chiara: la GUI non è solo un'interfaccia utente, ma anche un dispositivo di input per l'owner e di output per altri utenti (mirror).
- **Microservizi** Questo apre nuove possibilità, tra cui l'interazione machine-to-machine (M2M), grazie all'adozione delle WebSocket che ha permesso di immaginare un'evoluzione del sistema basata su microservizi, dove ogni componente può funzionare come un servizio indipendente e la GUI non è più un semplice frontend locale, ma un servizio indipendente che può interagire con altri componenti software.

Librerie

Per la realizzazione delle WebSocket, abbiamo considerato due approcci: utilizzare le funzioni standard della libreria WebSocket selezionata o adottare una libreria custom, unibo.basicomm23.

Il principale beneficio delle librerie custom è la standardizzazione dei processi: le funzioni complesse sono semplificate attraverso interfacce intuitive, facilitando l'uso. Questa libreria custom garantisce coerenza nello sviluppo, poiché consente di impiegare le stesse funzioni all'interno di un team, riducendo il rischio di errori “nascosti” associati all'uso diretto delle funzioni della libreria standard.

Ancora una volta, nascondere le libreria standard dietro la libreria Paho, semplifica l'interazione con MQTT, mantenendo il sistema modulare e riducendo la complessità per gli sviluppatori, garantendo nel contempo un'integrazione efficace con altri servizi.

Osservazioni

Il progetto di sviluppo di **ConwayGUI** ha rappresentato il mio primo approccio all'ingegneria del software, offrendomi un'importante opportunità per approcciare concetti fondamentali.
Durante questo progetto, ho avuto l'opportunità di imparare e lavorare con tecnologie come SpringBoot, comunicazioni via WebSocket, protocollo MQTT e Docker.
Questo approccio mi ha fornito una comprensione pratica di come gestire ambienti di sviluppo e produzione in modo efficiente. La combinazione di queste tecnologie ha ampliato le mie competenze e ha contribuito alla mia formazione nel campo dello sviluppo software.