Visione generale prima parte del corso:

- **conway25JavaScript, conway25Java**

Per realizzare una prima parte del gioco Conway Life siamo partiti con un approccio bottom-up, utilizzando JavaScript e poi Java per comprendere il funzionamento base (core business) del gioco.

Da questa versione Java abbiamo fatto il refactoring per introdurre una rappresentazione esplicita di Cell e Grid come classi Java, in generale durante tutto lo sviluppo si è basato sull'impostazione del prodotto in modo che possa evolvere tramite aggiunte progressive di funzionalità e senza modifiche (metodologia agile, sviluppo iterativo-incrementale, inversione delle dipendenze).

- **conwaygui**

Abbiamo poi realizzato una pagina HTML che funga da dispositivo di input-output evoluto per l'applicazione ConwayLife25 in Java, capace di interagire tramite WebSocket con un Server.

L'obiettivo era quello di realizzare la pagina HTML che comunica con il server modificando il meno possibile il codice già scritto, per questo abbiamo utilizzato il framework 'SpringBoot'.

- **commdemo**

La prima versione di interazione M2M inclusa nel progetto 'conwaygui' è stata la base per una forma più generale di comunicazione che abbiamo implementato nel progetto 'commdemo'.

Abbiamo quindi trasformato il prodotto, usando sempre SpringBoot, in un microservizio a sé stante, capace di comunicare con l'applicazione mediante scambio di messaggi usando MQT;

Il progetto commdemo: include esempi di interazione via WS e via MQTT.

- **conway25JavaMqtt, conwayguialone**

In questa seconda versione del sistema, usiamo SpringBoot per realizzare la GUI solo come 'dispositivo evoluto di I/O, in questa versione ci sono due agenti, attivati in modo indipendente e che interagiscono tra loro via MQTT.

conway25JavaMqtt => realizza una versione della logica del servizio, invia messaggi di aggiornamento dello stato di una cella sulla topic ***guiin*** e gestisce i comandi inviati dalla GUI sulla topic ***life.***

conwayguialone => nuova versione della GUI uguale alla precedente, ma non ha più il servizio applicativo embedded e usa la libreria '*Paho*' per la comunicazione via MQTT.

Aspetti significativi della Fase 1

1. ***Finalità prima parte e possibili finalità delle parti successive***:
   La prima fase del progetto si è concentrata sulla realizzazione incrementale del *Gioco della Vita* di Conway, con un approccio progressivo che parte da una versione locale fino a una più distribuita e complessa. Gli obiettivi principali includono:
   - Passaggio da JavaScript a Java.
   - Strutturazione del codice in modo modulare (separazione della logica di gioco dalla presentazione - GUI).
   - Conversione in microservizi, utilizzo di *Spring Boot* comunicazione con la GUI via MQTT per supportare interazioni in rete.

   Come abbiamo accennato nelle ultime lezioni le prossime fasi si concentrano su vari obiettivi:

   - Dagli Oggetti agli Attori, cioè cerchiamo di rendere ogni componente del sistema più autonomo e reattivo, utilizzeremo il linguaggio *qak* per la definizione e l'interazione tra attori.
   - Abbiamo parlato di Celle come Attori locali e distribuiti
   - L'obiettivo principale sarebbe quello di ottenere un sistema completamente distribuito basato su attori

2. ***Sistemi realizzati/sperimentati:***
   Durante questa prima fase del corso ho cercato di testare tutte le varie versioni dei progetti discussi, nella realizzazione mi sono concentrato sulla parte di refactoring iniziale e su alcune classi base delle versioni successive. Prendendo spunto dai progetti caricati dal professore e cercando di capirne il funzionamento (soprattutto le classi che si occupano della comunicazione tramite WS o MQTT).

3. ***Abilità/Competenze apprese:***
   Dal punto di vista degli aspetti concettuali compresi in questa fase sicuramente ci sono i vari principi dell'ingegneria del software, legati al modello di sviluppo incrementale:

   - **Divide et impera**: responsabilità distribuita fra i vari componenti (GUI,LifeController, ..)
   - **Singola responsabilità**: ogni classe ha un ruolo ben definito (manutenibilità).
   - **Disaccoppiamento**: Uso di interfacce per separare la logica di gioco dall'implementazione dell'output (IOutDev), si parla di inversione delle dipendenze.
   - **Modularità.**
   - **Refactoring.**
   - **Pattern MVC.**

   Dal punto di vista della parte del 'saper-fare' le competenze acquisite riguardano:
   - **Spring Boot**: Per trasformare l'applicazione in un microservizio, gestendo comunicazioni via HTTP e WebSocket.
   - **WebSocket**: Per la comunicazione bidirezionale tra server e client, per gli aggiornamenti dell'interfaccia.
   - **MQTT**: Protocollo di messaggistica utilizzato per l'invio di comandi e aggiornamenti tra componenti dell'applicazione (publish/subscrive).
   - **Gradle**: Gestione della build e delle dipendenze per lo sviluppo Java.

4. ***Motivo della scelta come caso di studio "Conway Life":***
   Secondo me è stato scelto questo caso di studio perché trami te regole semplici (nascita, sopravvivenza, morte delle celle), il comportamento e l'evoluzione del sistema sono complessi e imprevedibili (abbiamo parlato infatti di sistemi complessi).
   Inoltre, ci ha permesso di strutturare il progetto in fasi successive, partendo da una semplice implementazione locale fino ad arrivare a una versione distribuita e di esplorare varie tecnologie e metodologie.

5. ***Scelta di Java e framework legati a Java***
   Come abbiamo detto a lezione uno dei vantaggi principale di Java è la portabilità e compatibilità: il codice è eseguibile su qualsiasi piattaforma grazie alla JVM (soluzione ottima in un progetto basato su un'architettura distribuita).
   Inoltre, l'orientazione agli oggetti ci ha permesso di facilitare la modularizzazione e il disaccoppiamento fra componenti.
   L'utilizzo di framework come SpringBoot invece ci ha consentito di realizzare una pagina HTML per la GUI che interagisce con un server, tutto questo senza dover modificare il software già realizzato in Java.

6. ***Principi di Ingegneria del Software***: discussi nel punto 3.

7. ***Ruolo delle librerie 'custom':***

   Lo sviluppo di una libreria custom per MQTT ci ha permesso di rendere il sistema più modulare, mantenibile e scalabile, ci ha aiutato nella gestione della comunicazione tra i servizi.

   'unibo.basicomm23-1.0.jar' è stata utilizzata sia per standardizzare la gestione della comunicazione MQTT (per non dover implementare ogni volta la logica di connessione e scambio di messaggi) sia per separare la logica applicativa dalla gestione delle comunicazioni (la parte core funziona indipendentemente dal protocollo di comunicazione usato).