

Mastermind online: report finale

Andrea Negri - 0000990221
andrea.negri4@studio.unibo.it

Maggio 2023

Il progetto consiste nella realizzazione di una versione online del gioco da tavolo Mastermind¹. Ogni partita sarà formata da 2 giocatori, un codificatore e un decodificatore. Il codificatore sceglierà un codice segreto composto da 4 cifre decimali, che il decodificatore dovrà indovinare. Per ogni tentativo verranno forniti degli indizi sulla correttezza del codice. Il decodificatore avrà 9 tentativi per riuscire a indovinare il codice e quindi vincere, se non ci riuscirà avrà vinto il codificatore.

¹<https://it.wikipedia.org/wiki/Mastermind>

Indice

1	Obiettivo e Requisiti	3
1.1	Q/A	3
1.2	Scenari	3
1.3	Politiche di autovalidazione	4
2	Analisi	5
2.1	Requisiti di connettività	5
2.2	Requisiti di gioco	5
2.3	Requisiti di sistema	6
3	Design	6
3.1	Struttura	6
3.2	Comportamento	7
3.3	Interazioni	10
4	Dettagli dell'implementazione	11
4.1	client	11
4.2	common	12
4.3	server	12
4.4	test	12
5	Autovalidazione	13
6	Deployment	14
7	Esempi d'uso	15
8	Conclusioni	19
8.1	Sviluppi futuri	19
8.2	Cosa ho imparato	19

1 Obiettivo e Requisiti

1.1 Q/A

- *Cosa succede quando avvio l'applicazione?*
 - All'avvio dell'applicazione il giocatore potrà scegliere il proprio nickname (univoco all'interno del sistema) per poi accedere al menù principale.
- *Cosa posso fare dal menù principale?*
 - Una volta scelto il nickname, dal menù principale si potrà scegliere di creare una lobby o visualizzare le lobby esistenti e connettersi ad una di esse.
- *Cosa succede quando creo una lobby?*
 - Quando si sceglie di creare una lobby viene chiesto di scegliere anche il ruolo che si vorrà ricoprire (codificatore o decodificatore) dopodiché si rimane in attesa che un altro giocatore si unisca.
- *Cosa succede quando mi connetto ad una lobby?*
 - Quando un giocatore si connette ad una lobby assume in automatico il ruolo mancante. La partita ha quindi inizio.
- *Come si svolge una partita?*
 - Una partita si svolge esclusivamente tra 2 giocatori, un codificatore e un decodificatore. A inizio partita il codificatore sceglie un codice segreto composto da 4 cifre decimali (compreso quindi tra 0000 e 9999). Gli zero sono **tutti** importanti anche se a sinistra della prima cifra significativa (0001 non verrà abbreviato in 1). Ad ogni turno il decodificatore effettua un tentativo per indovinare il codice segreto. Prese le 4 cifre del tentativo viene quindi comunicato al decodificatore il **numero** di cifre giuste al posto giusto e il **numero** di cifre giuste al posto sbagliato. *Es. con codice 3465, se il decodificatore prova con 1457, gli verrà comunicata la presenza di una cifra giusta al posto giusto(4), e una cifra giusta al posto sbagliato(5).* Il decodificatore ha 9 tentativi per indovinare il codice segreto: se ci riesce ha vinto. Altrimenti vince il codificatore.

1.2 Scenari

Nelle Figure 1 e 2 sono riportati i diagrammi dei casi d'uso dal punto di vista delle funzionalità di connessione e di gioco utilizzabili dal giocatore. In Figura 1 vengono mostrate le funzionalità di connessione: un giocatore può registrarsi inserendo un nickname e diventare così un giocatore registrato oppure disconnettersi. Un giocatore registrato è un giocatore che, come detto precedentemente, ha inserito un nickname e può quindi decidere di creare una lobby (scegliendo il ruolo che vuole giocare) oppure unirsi a una lobby esistente (ottenendo il ruolo mancante nella lobby).

Una volta che in una lobby sono presenti 2 giocatori, la partita ha inizio, e gli scenari possibili sono rappresentati in Figura 2. Se il giocatore è il codificatore dovrà scegliere un codice di 4 cifre decimali, dopodiché non dovrà far altro che aspettare i tentativi del decodificatore. Se invece il giocatore è il decodificatore dovrà aspettare che il codificatore scelga il codice e poi avrà 9 tentativi per provare a indovinarlo. Prese le 4 cifre del tentativo, gli verranno comunicati il **numero** di cifre giuste al posto giusto e il **numero** di cifre giuste al posto sbagliato.

La partita ha fine quando il decodificatore indovina il codice (ed è il vincitore) oppure quando finisce i tentativi (in questo vince il codificatore).

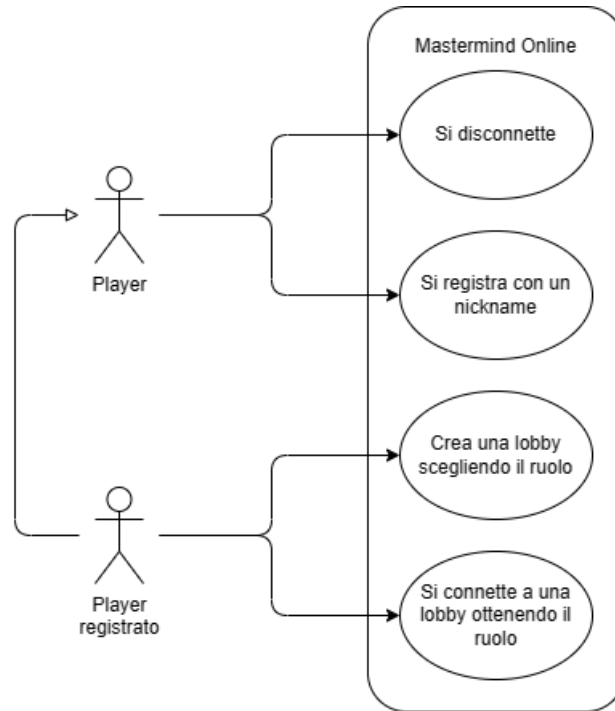


Figura 1: Diagramma casi d'uso delle funzionalità di connessione

1.3 Politiche di autovalidazione

Il progetto comprenderà un insieme di test automatici implementati tramite il framework JUnit, che consentiranno di verificare sia il corretto funzionamento dei componenti presi singolarmente che integrati tra loro (quindi con la simulazione della logica di una partita).

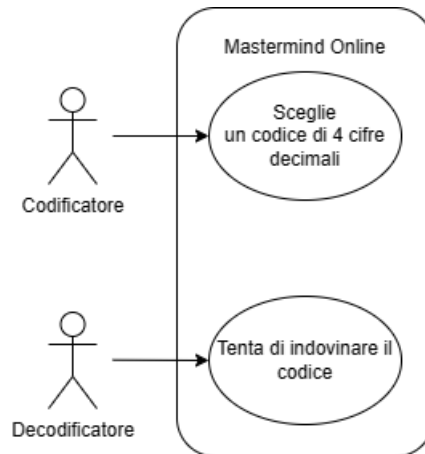


Figura 2: Diagramma casi d'uso delle funzionalità di gioco

2 Analisi

Alla luce di quanto descritto nel precedente capitolo i requisiti si possono dividere in **requisiti di connettività** e **requisiti di gioco**, inoltre si possono ora definire anche i **requisiti di sistema**.

2.1 Requisiti di connettività

- Il giocatore deve poter inserire un nickname quando avvia l'applicazione
- Il giocatore deve quindi poter scegliere se creare una lobby o unirsi a una già esistente
- Se decide di creare una lobby deve poter scegliere il ruolo da ricoprire
- Se decide di unirsi a una lobby gli dovrà essere assegnato il ruolo mancante
- La partita ha inizio quando nella lobby sono presenti 2 giocatori
- Al termine della partita il giocatore può nuovamente decidere se giocare o uscire

2.2 Requisiti di gioco

- A inizio partita il codificatore sceglierà un codice segreto seguendo le regole descritte precedentemente nella sezione 1.1
- Una volta scelto il codice, il decodificatore potrà iniziare a tentare di indovinarlo, sempre seguendo le regole descritte nella sezione 1.1
- Il decodificatore ha 9 tentativi per indovinare il codice segreto: se ci riesce ha vinto. Altrimenti vince il codificatore

2.3 Requisiti di sistema

- L'architettura scelta per l'implementazione del sistema è quella client-server
- Il server verrà sviluppato come Web Service dotato di API ReST, utilizzando la libreria `io.javalin.Javalin` ², e `com.google.code.gson` ³ per la serializzazione e deserializzazione
- Il server si occuperà di gestire le lobby e le relative connessioni dei client
- I client utilizzeranno una CLI con la quale gli utenti potranno utilizzare il sistema

3 Design

3.1 Struttura

In Figura 3 viene mostrato il diagramma delle classi del Model (non sono presenti getter e setter). Vediamo ora nel dettaglio i componenti:

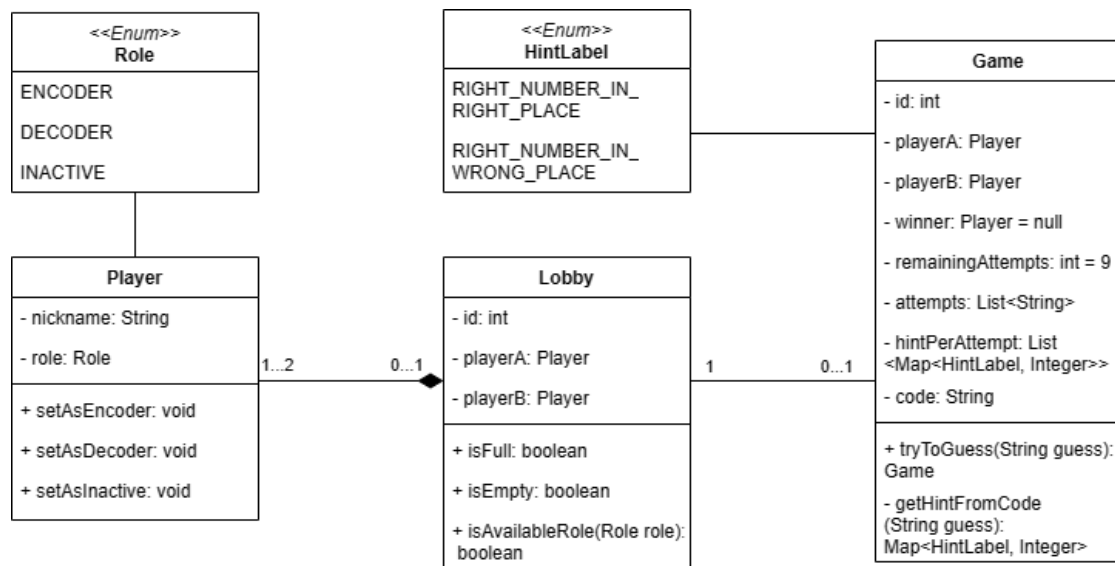


Figura 3: Diagramma delle classi del model

- **Role**: questo enum indica il ruolo che ricopre attualmente il giocatore. **ENCODER** e **DECODER** indicano rispettivamente codificatore e decodificatore e vengono impostati quando un giocatore crea una lobby o quando vi entra. Il ruolo **INACTIVE** indica un giocatore che non sta giocando (non è connesso a una lobby) per cui non ha un ruolo assegnato;

²<https://javalin.io/>

³<https://github.com/google/gson>

- **HintLabel:** l'enum in questione viene utilizzato dopo ogni tentativo del decodificatore, per salvare e mostrare il numero di cifre giuste al posto giusto (**RIGHT NUMBER IN RIGHT PLACE**) e il numero di cifre giuste al posto sbagliato (**RIGHT NUMBER IN WRONG PLACE**);
- **Player:** rappresenta il giocatore, con relativo *nickname* (univo all'interno del sistema) e *role* (preso dall'enum Role). I metodi mostrati (*setAsEncoder*, *setAsDecoder*, *setAsInactive*) servono a impostare il ruolo del giocatore;
- **Lobby:** rappresenta una lobby alla quale possono connettersi 2 giocatori per poi giocare una partita. È composta da un *id*, e dai 2 giocatori, *playerA* e *playerB*. Quando viene creata solo uno dei 2 giocatori è presente (quello che crea la lobby), l'altro verrà impostato quando si conatterà un giocatore. I metodi presenti consentono di verificare se la lobby è piena (*isFull*, sono presenti 2 giocatori), vuota (*isEmpty*, non è presente alcun giocatore) o verificare se un determinato ruolo è disponibile (*isAvailableRole(Role role)* che, dato un ruolo, verifica se è già presente o meno in lobby);
- **Game:** infine, l'entità più importante, quella che rappresenta una partita. Al suo interno sono presenti un *id*, i giocatori (*playerA* e *playerB*), *winner*, che indica il vincitore, *code*, il codice segreto da indovinare, *remainingAttempts*, il numero di tentavi rimanenti al decodificatore, *attempts*, la lista di tentativi effettuati, e *hintPerAttempt*, che contiene i 2 indizi per ogni tentativo effettuato. Il metodo pubblico *tryToGuess(String guess)* consente di effettuare un tentativo e al suo interno richiama il metodo *getHintFromCode(String guess)* che, dato il tentativo, restituisce i 2 indizi.

3.2 Comportamento

In Figura 4 viene mostrato il diagramma degli stati relativo al comportamento della lobby in risposta agli input dell'utente. Per prima cosa viene richiesto di inserire un nickname e in seguito viene mostrato un menù dal quale, come detto precedentemente, un giocatore può scegliere di creare una lobby, visualizzare le lobby disponibili o uscire dal gioco. Se si sceglie di creare una lobby viene richiesto di scegliere in che ruolo giocare e poi il giocatore dovrà attendere che un altro giocatore si connetta. Se invece si sceglie di visualizzare le lobby disponibili, il giocatore potrà decidere a quale di queste connettersi oppure tornare al menù principale. Se si è in attesa che un giocatore si connetta e questo accade, oppure se si decide di unirsi a una delle lobby presenti, la partita ha inizio. A fine partita si viene riportati alla visualizzazione del menù principale.

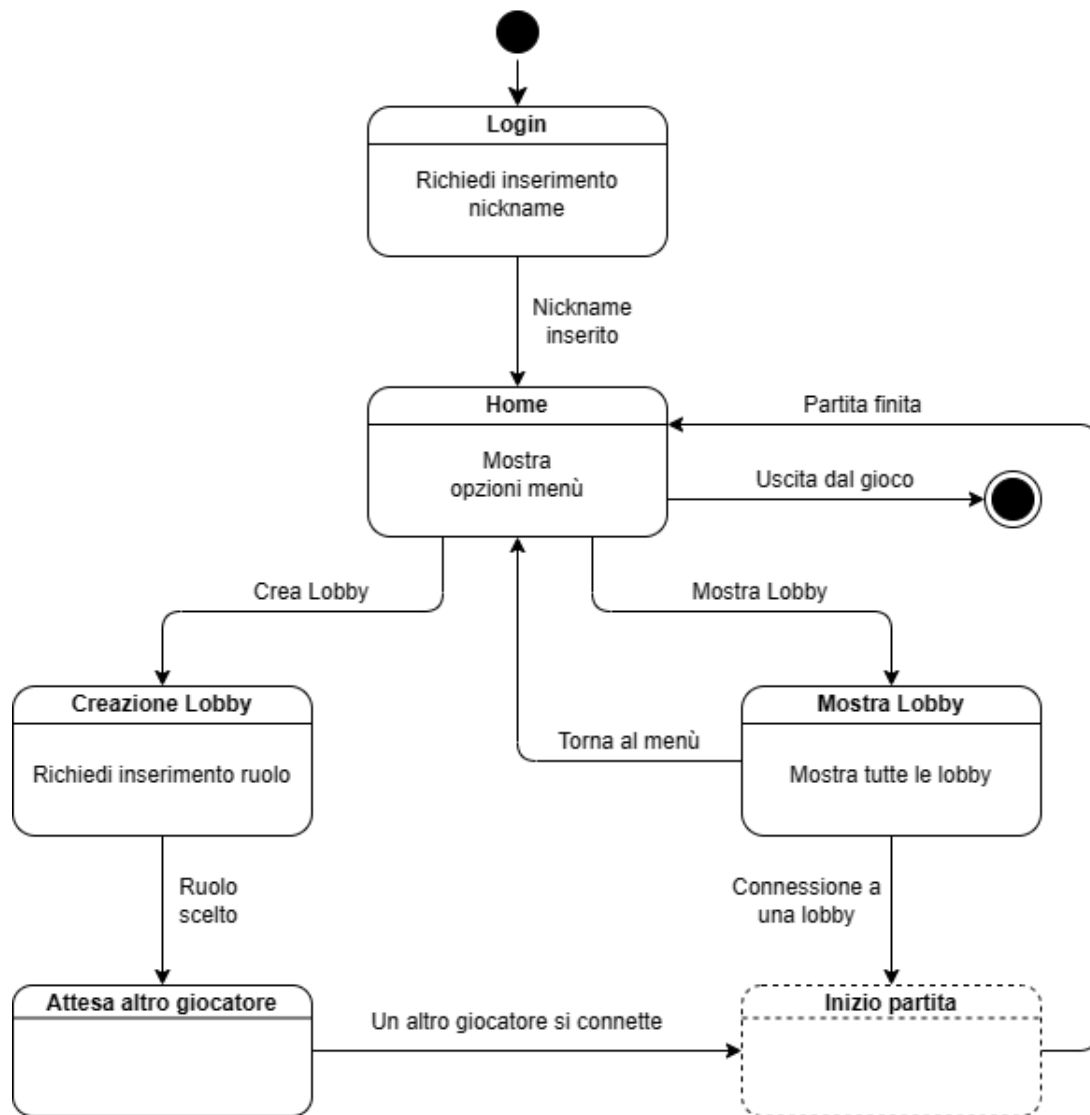


Figura 4: Diagramma degli stati della lobby

Il diagramma degli stati relativo alla partita viene mostrato in Figura 5. A seconda del ruolo ricoperto dal giocatore cambiano gli stati:

- **codificatore (encoder)**: deve innanzitutto scegliere il codice segreto da far indovinare. Una volta scelto il codice non dovrà far nient'altro ma visualizzerà i tentativi effettuati dal decodificatore, e gli verrà comunicato il risultato della partita: vittoria in caso il decodificatore abbia finito i tentativi senza indovinare, sconfitta nel caso in cui il decodificatore abbia indovinato il codice.
- **decodificatore (decoder)**: all'inizio dovrà aspettare che il codificatore scelga il

codice. Una volta scelto il codice potrà iniziare a tentare di indovinarlo. Per ogni tentativo errato gli verranno mostrati gli indizi, cioè il numero di cifre corrette nei posti giusti e nei posti sbagliati, oltre al numero di tentativi rimasti. Se indovina il codice vince la partita se invece termina i tentativi la perde.

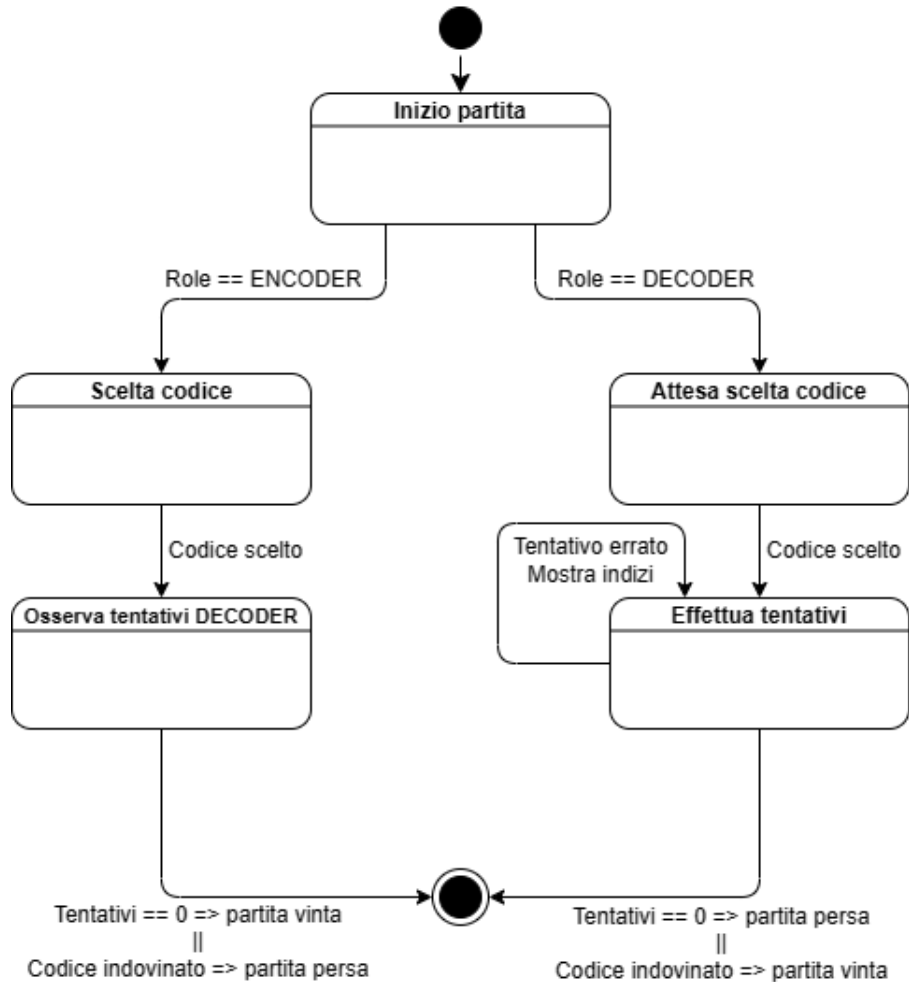


Figura 5: Diagramma degli stati della partita

Durante la partita sono impostati dei timer, sia per il codificatore che per il decodificatore, per poter gestire i casi di eventuali disconnessioni. In particolare:

- il codificatore ha 45 secondi per impostare il codice. Se non lo fa, il decodificatore si disconnette dalla partita. Il codificatore a questo punto rimane solo in lobby e una volta scelto il codice, gli verrà comunicato che è solo in partita e verrà riportato al menù principale;
- una volta scelto il codice, il decodificatore ha 45 secondi per effettuare un tentativo, se non lo fa il codificatore si disconnette. A questo punto il decodificatore rimane

solo in lobby e dopo aver effettuato il tentativo verrà avvertito che è rimasto solo in partita e verrà riportato al menù principale.

Il timer così impostato ha un duplice scopo: evitare che un giocatore rimanga in attesa infinita della mossa dell'altro e gestire i casi in cui uno dei 2 giocatori si disconnette in maniera improvvisa e non voluta.

Inoltre, per gestire casi di disconnessioni non volute (disconnessioni non graceful) è stato implementato un meccanismo di heartbeat lato server: nel momento in cui un giocatore si connette al server, il relativo client inizia a inviare un segnale ogni 5 secondi per comunicare che è ancora online. Il server tiene quindi traccia dello stato di connettività dei vari client grazie ai segnali ricevuti: se non riceve alcun segnale da un determinato client per 10 secondi lo considera disconnesso e quindi provvede a eliminare il giocatore (rendendo disponibile nuovamente il nickname), eventualmente anche a disconnetterlo dalla lobby a cui era connesso.

3.3 Interazioni

Lo scambio di informazioni tra client e server avviene tramite varie richieste HTTP e la libreria **Gson** che consente di serializzare oggetti Java nella loro rappresentazione JSON e viceversa, deserializzarli. In Figura 6. viene mostrato un diagramma di sequenza nel quale si possono osservare le interazioni svolte per:

- creare una lobby: **RemoteMastermind1** effettua una chiamata HTTP POST per creare una lobby e riceve come risposta la lobby creata;
- visualizzare le lobby disponibili: **RemoteMastermind2** effettua una chiamata HTTP GET per visualizzare le lobby disponibili e come risposta riceve una lista contenente le lobby;
- unirsi a una lobby: **RemoteMastermind2** effettua una chiamata HTTP PUT per unirsi a una lobby e riceve come risposta la lobby a cui si è connesso.

In maniera simile si svolgono anche le altre interazioni riguardanti **Player** e **Game**. Si tenga conto che **RemoteMastermind** (entrambi) e **LocalMastermind** implementano la stessa interfaccia **Mastermind** che fornisce i metodi necessari a gestire lobby, giocatori e partite. La differenza è che il primo viene utilizzato lato client e i metodi implementati effettuano delle richieste HTTP che vengono inviate al **MastermindService** il quale, utilizza il secondo, che invece si occupa di gestire effettivamente lobby, giocatori e partite.

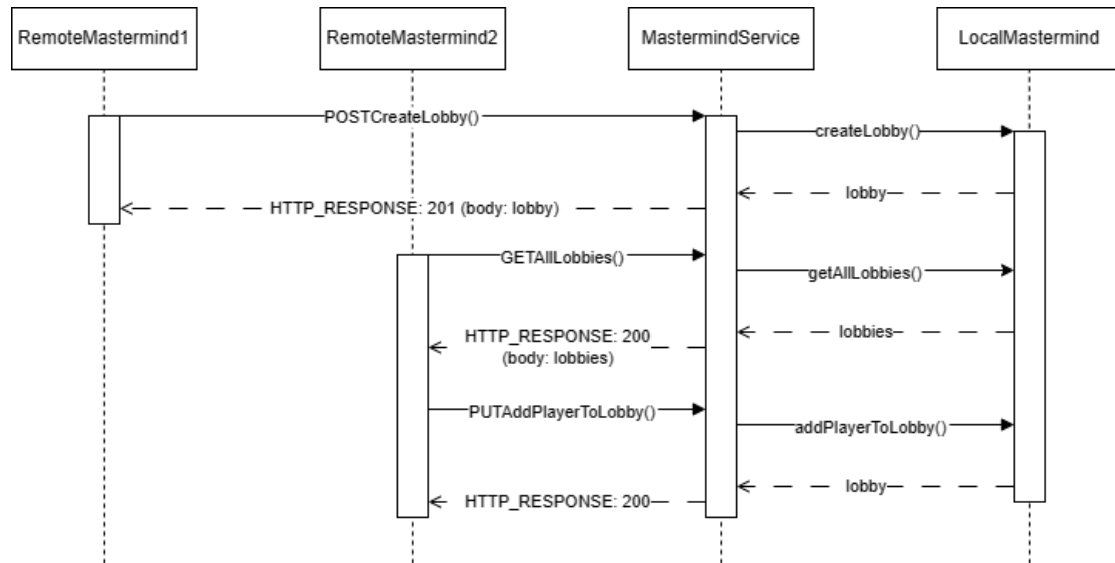


Figura 6: Diagramma di sequenza per la creazione, visualizzazione e connessione di una lobby

4 Dettagli dell'implementazione

Vengono ora illustrati alcuni dettagli dell'implementazione. A livello di tecnologie, il progetto è stato implementato in linguaggio **Java**, e utilizzando **Gradle** come build automation tool. I moduli presenti nel progetto sono:

- client
- common
- server
- test

4.1 client

Il modulo client contiene 2 componenti principali:

- **MastermindClient**: gestisce l'interazione con l'utente tramite la CLI;
- **RemoteMastermind**: è l'implementazione dell'interfaccia Mastermind utilizzata da ogni client. Essa viene richiamata dal MastermindClient per effettuare le chiamate necessarie al server. Ogni metodo è implementato tramite una chiamata HTTP al server (l'elenco viene mostrato nella sezione 4.3).

4.2 common

Il modulo common contiene gli elementi del modello (mostrati nella sezione 3.1), l'interfaccia **Mastermind** e la sua implementazione locale (**LocalMastermind**), oltre che ai serializzatori e deserializzatori dei vari elementi del modello, scritti tramite l'ausilio di Gson.

4.3 server

Il server è stato realizzato tramite **Javalin** che ha consentito di creare un Web Service dotato di Api ReST. Esso lavora su 3 risorse (Game, Lobby e Player) per ognuna delle quali è stato definito un Controller e un Api. Nel controller sono definite le rotte (URL) disponibili per interagire con ogni risorsa, e l'interazione vera e propria avviene grazie alla relativa Api. Ogni Api interagisce con la versione locale del gioco, cioè **LocalMastermind**. Le rotte sono state documentate tramite l'ausilio di Swagger⁴ e sono mostrate nelle Figure 7, 8 e 9.

games	
POST	/mastermind/v0.1.0/games/{id}
DELETE	/mastermind/v0.1.0/games/{id}
GET	/mastermind/v0.1.0/games/{id}
PUT	/mastermind/v0.1.0/games/{id}/{name}
PUT	/mastermind/v0.1.0/games/attempt/{id}/{name}

Figura 7: Api per le partite

4.4 test

All'interno di questo modulo sono presenti vari test che verificano la correttezza della serializzazione, deserializzazione, della logica delle partite (sia in locale che tramite rete), della gestione delle lobby e dei giocatori (sia in locale che tramite rete).

⁴<https://swagger.io/>

lobbies	
POST	/mastermind/v0.1.0/lobbies
GET	/mastermind/v0.1.0/lobbies
DELETE	/mastermind/v0.1.0/lobbies/{id}
GET	/mastermind/v0.1.0/lobbies/{id}
PUT	/mastermind/v0.1.0/lobbies/{id}/{name}/{role}
DELETE	/mastermind/v0.1.0/lobbies/{id}/{name}

Figura 8: Api per le lobby

players	
POST	/mastermind/v0.1.0/players/{name}
DELETE	/mastermind/v0.1.0/players/{name}
GET	/mastermind/v0.1.0/players/{name}

Figura 9: Api per i giocatori

5 Autovalidazione

Come detto nella sezione precedente, all'interno del progetto è presente un modulo dedicato esclusivamente ai test, implementati tramite JUnit 5. I test creati prevedono:

- la verifica della corretta serializzazione e deserializzazione degli oggetti Java (Player, Lobby e Game);
- la verifica della corretta gestione dei giocatori;
- la verifica della corretta gestione delle lobby;
- la verifica della corretta gestione delle partite e dell'intera logica di gioco.

Per gli ultimi 3 punti si è provveduto a creare sia test che vengono eseguiti in "ambiente locale" cioè che **non** utilizzano né un server né chiamate HTTP (l'unico componente coinvolto è il LocalMastermind), sia in un ambiente di rete, con server e relative chiamate HTTP (si coinvolge quindi un LocalMastermind e un MastermindService per il server e un RemoteMastermind per il client).

Sono state inoltre sfruttate le *CI/CD Pipeline* di GitLab prendendo spunto da quanto visto durante i laboratori affinché, ad ogni push, venissero eseguiti in automatico i test presenti. Il contenuto del file `.gitlab-ci.yml` tramite il quale si è configurata l'automatizzazione dei test è mostrato di seguito:

```
image: gradle:latest

stages:
  - build
  - test

build-job:
  stage: build
  script:
    - cd mastermind-online
    - gradle classes testClasses

unit-test-job:
  stage: test
  script:
    - cd mastermind-online
    - gradle test
  artifacts:
    when: always
    reports:
      junit: '**/build/test-results/test/**/TEST-*.xml'
```

6 Deployment

Per eseguire il progetto basta utilizzare una macchina dotata di JVM (con versione ≥ 17). Per prima cosa bisogna posizionarsi con il terminale all'interno della cartella *mastermind* (quella contenente il progetto), dopodiché avviare il server con il comando

```
./gradlew server:run
```

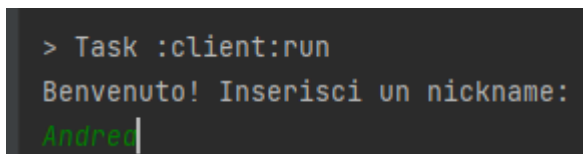
In questa maniera è stato avviato il server su `http://localhost:10000` (la porta si può cambiare, passandola come argomento al comando specificato sopra). Per lanciare i client invece, dalla stessa directory, si può utilizzare il comando

```
./gradlew --console plain client:run
```

L'opzione `--console plain` evita di visualizzare i messaggi di gradle, e per avviare più client basta lanciare il comando più volte in finestre diverse.

7 Esempi d'uso

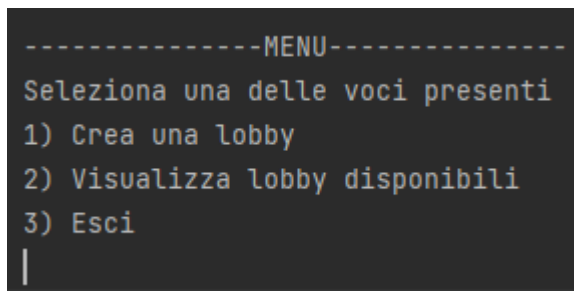
Per prima cosa bisogna avviare il server come mostrato sopra. Fatto questo si può procedere ad avviare un client che richiederà l'inserimento di un nickname (Figura 10).



```
> Task :client:run
Benvenuto! Inserisci un nickname:
Andrea|
```

Figura 10: Inserimento nickname

Inserito il nickname verrà visualizzato il menù principale (Figura 11).



```
-----MENU-----
Seleziona una delle voci presenti
1) Crea una lobby
2) Visualizza lobby disponibili
3) Esci
|
```

Figura 11: Menù principale

Se si sceglie di creare una lobby, verrà chiesto in quale ruolo si vuole giocare e si viene quindi messi in attesa che un altro giocatore si connetta (Figura 12).

```
1

Seleziona un ruolo:
1) Codificatore
2) Decodificatore
1

-----
Lobby 1 creata.
In attesa di un altro giocatore...
|
```

Figura 12: Creazione lobby

Se invece sei sceglie di visualizzare le lobby disponibili esse vengono visualizzate come mostrato in Figura 13.

```
2

-----LOBBIES-----
Lobby 1 - Ruolo mancante: DECODER - Player collegato: Andrea

R) Refresh lista
B) Torna al menu
Inserisci codice lobby a cui connettersi: |
```

Figura 13: Visualizzazione lobby

Quando in una lobby sono connessi 2 giocatori ha inizio la partita: il codificatore deve scegliere il codice (Figura 14) mentre il decodificatore viene messo in attesa (Figura 15).


```
-----INIZIO GAME-----  
Sei il codificatore  
Inserisci il codice da far indovinare (ricorda, deve essere composto da 4 cifre DECIMALI (0-9):  
1463|
```

Figura 14: Il codificatore sceglie il codice

```
-----INIZIO GAME-----  
Sei il decodificatore  
Attendi che il codificatore scelga un codice...
```

Figura 15: Il decodificatore attende

Una volta scelto il codice il codificatore visualizzerà i tentativi del decodificatore (Figura 16) che invece tenterà di indovinarlo e di volta in volta visualizzerà gli indizi che gli vengono forniti(Figura 17).

```
Codice 1463 impostato  
Il decodificatore ha provato il codice 4523
```

Figura 16: Il codificatore visualizza i tentativi dell'altro giocatore

```
Il codificatore ha scelto il codice  
Tentativi rimasti: 9  
Inserisci il tuo tentativo (ricorda, deve essere composto da 4 cifre DECIMALI (0-9):  
4523  
Hai provato il codice 4523  
Cifre giuste al posto giusto: 1  
Cifre giuste al posto sbagliato: 1  
-----  
Tentativi rimasti: 8  
Inserisci il tuo tentativo (ricorda, deve essere composto da 4 cifre DECIMALI (0-9):
```

Figura 17: Il decodificatore prova a indovinare il codice

Quando il codice viene indovinato il codificatore perde (Figura 18) mentre il decodificatore vince (Figura 19).

```
Mi dispiace, hai perso :(
```

Figura 18: Il codificatore ha perso

```
Tentativi rimasti: 8  
Inserisci il tuo tentativo (ricorda, deve essere composto da 4 cifre DECIMALI (0-9):  
1483  
COMPLIMENTI! HAI VINTO!
```

Figura 19: Il decodificatore ha vinto

Se invece finisco i tentativi, vince il codificatore (Figura 20) e perde il decodificatore (Figura 21).

```
Codice 1973 impostato  
Il decodificatore ha provato il codice 4564  
Il decodificatore ha provato il codice 7897  
Il decodificatore ha provato il codice 1232  
Il decodificatore ha provato il codice 4562  
Il decodificatore ha provato il codice 3258  
Il decodificatore ha provato il codice 1594  
Il decodificatore ha provato il codice 3575  
Il decodificatore ha provato il codice 4568  
COMPLIMENTI! HAI VINTO!
```

Figura 20: Il codificatore ha vinto

```
Tentativi rimasti: 1  
Inserisci il tuo tentativo (ricorda, deve essere composto da 4 cifre DECIMALI (0-9):  
1595  
Mi dispiace, hai perso :(
```

Figura 21: Il decodificatore ha perso

8 Conclusioni

Con questo progetto è stata realizzata una versione online del gioco Mastermind, facilmente utilizzabile da chiunque grazie all'utilizzo di Gradle che consente di poter eseguire l'applicazione senza particolari procedure preliminari.

8.1 Sviluppi futuri

Tra le possibili feature da implementare ci sono:

- fornire meccanismi di registrazione e autenticazione più avanzati, supportati da un apposito database, grazie ai quali si potrebbe tener traccia delle informazioni degli utenti, come ad esempio il loro storico delle partite, e delle relative statistiche;
- fornire la possibilità di filtrare le lobby per ruolo, in maniera tale da visualizzare solo quelle con il ruolo desiderato;
- creare una GUI, sicuramente più user-friendly rispetto alla CLI.

8.2 Cosa ho imparato

Il progetto mi ha permesso di sperimentare l'utilizzo di alcune tecnologie che non avevo mai visto prima come **Gson** e **Javalin**, approfondire argomenti che già conoscevo con la pratica come per l'implementazione delle Api ReST e di un Web Service.

Soprattutto, però, ho reputato molto utile, sia per il presente che per il futuro, utilizzare un tool come Gradle che avevo finora visto solo superficialmente e, anche se effettivamente hanno occupato un tempo decisamente limitato del progetto, pure la creazione delle Pipeline su GitLab mi è stata molto utile. Proverò sicuramente a utilizzare questi 2 strumenti (se non loro, qualcosa di equivalente come può essere sbt, npm per Gradle e le Github Actions per le Pipeline) anche nei miei lavori futuri.