TPSIT – Progettazione

Descrizione del progetto

Per questo progetto ho deciso di realizzare un gioco multiplayer (client/server) chiamato: “RPS Advanced” (“Rock Paper Scissors Advanced”).

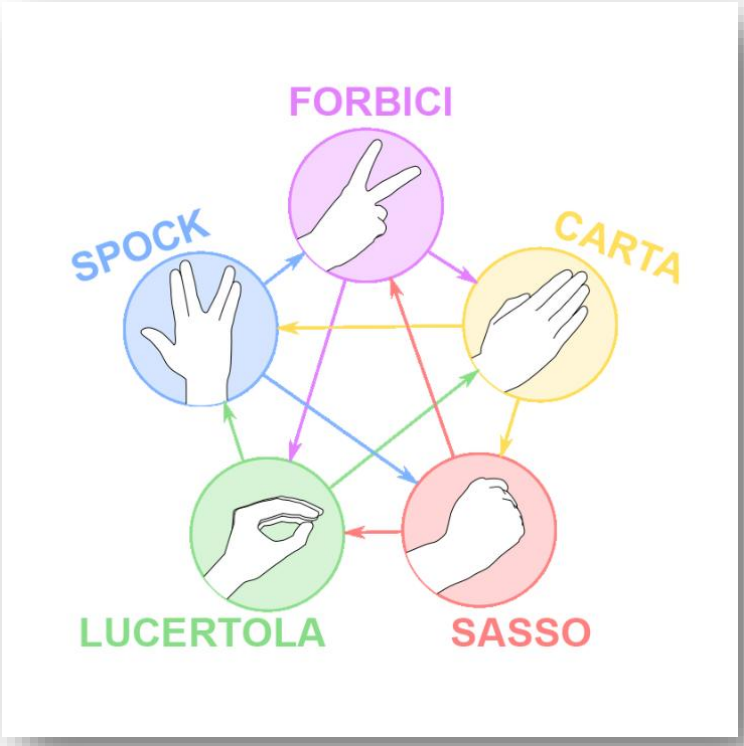
Questo gioco è un avanzamento del tradizionale gioco “sasso carta forbice”, infatti vengono aggiunti due nuovi elementi da giocare: la lucertola e Spock (Spock è un personaggio immaginario del franchise di fantascienza *“Star Trek”*). Questo “miglioramento” del gioco tradizionale è preso dalla serie TV: *“The Big Bang Theory”*. Per più informazioni è possibile vedere la pagina ufficiale del gioco sulla wiki Fandom della serie QUI.

Regole:

Il gioco è un'espansione del gioco “Sasso, Carta, Forbice” con i segni delle mani aggiuntivi della Lucertola (simile a una marionetta) e di Spock (il saluto vulcaniano della serie “Star Trek”). Ogni giocatore sceglie una variabile e la rivela contemporaneamente. Il vincitore è colui che sconfigge gli altri. In caso di parità, il processo viene ripetuto finché non viene trovato un vincitore.

Ecco le combinazioni possibili (pareggi esclusi):

Mossa	Batte	Viene battuta da
Sasso	Forbice, Lucertola	Carta, Spock
Carta	Sasso, Spock	Forbice, Lucertola
Forbice	Carta, Lucertola	Sasso, Spock
Lucertola	Carta, Spock	Sasso, Forbice
Spock	Sasso, Forbice	Carta, Lucertola



Progettazione

1. Linguaggio di Programmazione

Il linguaggio di programmazione scelto è Java per questo progetto, poiché ho già conoscenza e "confidenza" con questo linguaggio e con l'utilizzo di Socket e programmazione client/server con concorrenza con esso.

2. Librerie per la Comunicazione di Rete

Poiché il sistema deve utilizzare i Socket, possiamo gestire la comunicazione con la seguente classe nativa di Java:

- `java.net.Socket` e `java.net.ServerSocket` per il protocollo TCP.

Poiché i dati vengono trasmessi in formato JSON, userò una libreria per la conversione tra oggetti Java e JSON:

- Jackson (`com.fasterxml.jackson.core:jackson-databind`) per un JSON parsing rapido e flessibile.

Per il debug e la gestione dei log:

- SLF4J + Logback (`ch.qos.logback:logback-classic`) per un Logging strutturato.

3. Funzionalità Utente

- **Funzionalità essenziali:**
  - Creazione e gestione di partite multiplayer.
  - Invio e ricezione delle mosse tra i giocatori.
  - Regole di gioco.
  - Notifica della vittoria o sconfitta a fine partita.
  - Gestione della comunicazione tra client e server.

- **Funzionalità opzionali:**
  - Storico delle partite con risultati.
  - Classifica giocatori basata sulle vittorie.
  - Supporto per più modalità di gioco (es. modalità torneo).
  - Log delle partite salvato per revisione/debug.

4. Protocollo di Rete

Il protocollo di rete scelto è il **TCP** (Transmission Control Protocol):

- Affidabile (garantisce che i pacchetti arrivino e nel giusto ordine).
- Maggiore overhead (gestione connessioni, controlli di errore).
- Indicato per scambi strutturati come le mosse del gioco.

Il gioco non necessita di latenza ultra-bassa, ma richiede l'affidabilità nella ricezione delle mosse. L'uso di UDP avrebbe senso solo se il gioco fosse rapidissimo e accettasse perdita di pacchetti, cosa non applicabile a un gioco a turni come questo.

Per una gestione più avanzata delle connessioni (es. timeout, handling errori avanzato), userò Apache Commons IO (`commons-io:commons-io`) per semplificare la gestione di stream di input/output.

5. Sicurezza e Crittografia

- **Cosa criptare:**
  - Le mosse dei giocatori prima della trasmissione per evitare il man-in-the-middle (MITM).
  - I risultati della partita per proteggere l'integrità del gioco.
  - Le credenziali del giocatore (per una opzionale autenticazione).

- **Quando criptare:**
  - Prima dell'invio attraverso il Socket.
  - Decrittare solo all’arrivo nel lato corretto (server o client).

- **Algoritmi presi in considerazione:**
  - AES-256 (Advanced Encryption Standard) per crittografia simmetrica sui dati sensibili.
  - RSA-2048 o superiore per lo scambio sicuro delle chiavi AES tra client e server.
  - HMAC-SHA256 per garantire l'integrità dei messaggi.

- **Scambio chiavi:**
  - Il server genera una coppia di chiavi RSA e distribuisce la chiave pubblica ai client.
  - I client cifrano una chiave AES casuale con la chiave pubblica del server.
  - Il server decifra la chiave AES e la usa per la comunicazione crittografata.

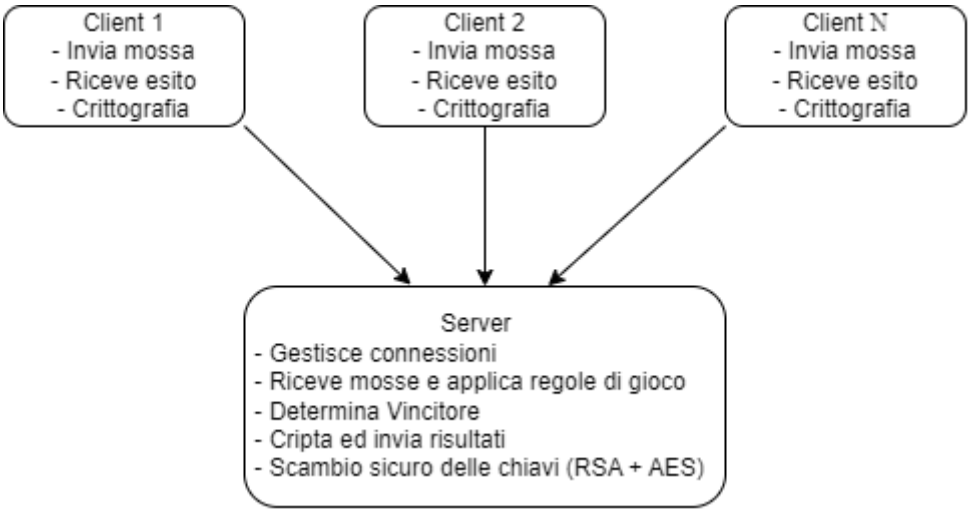
Architettura del Sistema

1. Client

- Invia le mosse al server.
- Riceve le risposte e le mostra all'utente.
- Cifra i dati sensibili prima di inviarli.

2. Server

- Gestisce le connessioni dei client.
- Applica le regole del gioco.
- Determina il vincitore e invia i risultati.
- Decifra i messaggi ricevuti dai client.



- **Flusso di dati:**
  - Connessione: i client si connettono al server via TCP Socket.
  - Scambio chiavi: il server invia la chiave pubblica RSA ai client, i client generano una chiave AES e la cifrano con RSA.
  - Invio mosse: i client inviano le mosse crittografate con AES.
  - Elaborazione: il server applica le regole di gioco e determina il vincitore.
  - Risultato: il server invia il risultato cifrato ai client.

- **Formato dei messaggi:**

Ogni messaggio sarà rappresentato come una stringa in formato JSON per facilitare la serializzazione e deserializzazione.

Tipi di messaggi

Tipo	Descrizione	Campi JSON
`HELLO`	Il client si connette al server e richiede la chiave pubblica	`{ "type": "HELLO", "username": "NomeGiocatore" }`
`PUBLIC_KEY`	Il server risponde con la chiave pubblica RSA	`{ "type": "PUBLIC_KEY", "key": "base64_RSA_key" }`
`AES_KEY`	Il client invia la chiave AES criptata con RSA	`{ "type": "AES_KEY", "key": "base64_AES_key_encrypted" }`
`MOVE`	Il client invia la mossa crittografata con AES	`{ "type": "MOVE", "data": "base64_mossa_encrypted" }`
`RESULT`	Il server invia il risultato della partita	`{ "type": "RESULT", "data": "base64_result_encrypted" }`

- **Flusso della Comunicazione**

1. **Handshake e Scambio di Chiavi**
   - Il client invia `HELLO` con il suo username.
   - Il server risponde con `PUBLIC_KEY` contenente la chiave RSA.
   - Il client genera una chiave AES casuale, la cifra con RSA e la invia come `AES_KEY`.

2. **Scambio delle Mosse**
   - Il client invia la propria mossa criptata con AES (`MOVE`).
   - Il server riceve e memorizza la mossa.

3. **Elaborazione e Risultato**
   - Quando entrambi i giocatori hanno inviato la loro mossa, il server determina il vincitore.
   - Il risultato viene inviato ai client come `RESULT`, crittografato con AES.

4. **Gestione delle Connessioni**
   - Il server gestisce più client con thread separati.
   - Ogni client ha il proprio Socket per la comunicazione.
   - Le mosse vengono gestite in modo sincrono per garantire che entrambi i giocatori abbiano inviato la loro mossa prima di determinare il vincitore.