Documentazione del protocollo di comunicazione

Leonardo Bianconi, Alexandru Gabriel Bradatan, Mattia Busso
Gruppo 36

29 aprile 2022

Indice

1 Introduzione

In questo documento verrà dettagliato il protocollo di comunicazione tra client e server dell'implementazione di Eriantys del gruppo 36.

Il protocollo userà JSON e sarà codificato in UTF-8.

2 Sequenza

Distinguiamo 5 possibili scenari di interazione tra le parti:

1. Ricerca delle partite

2. Creazione di una nuova partita
3. Unione/abbandono di una partita
4. Fase di gioco
5. Ping

Tranne per il ping, che è continuo dall'unione del giocatore fino alla fine della partita, tutti gli scenari si svolgeranno nella sequenza con cui sono stati indicati sopra.

2.1 Ricerca delle partite

Il client manderà una richiesta di tipo `FETCH` al server, il quale risponderà con una lista delle lobby non ancora al completo in una risposta di tipo `LOBBIES`.

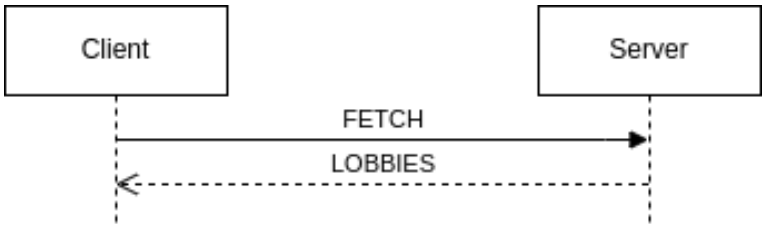

Figura 1: Scenario di ricerca delle partite

2.2 Creazione di una partita

Se un giocatore volesse creare una nuova lobby, il client manderà al server un messaggio di tipo `CREATE`. Il server risponderà con un messaggio di tipo `UPDATE` che conterrà lo stato di default di una partita non ancora inizializzata.

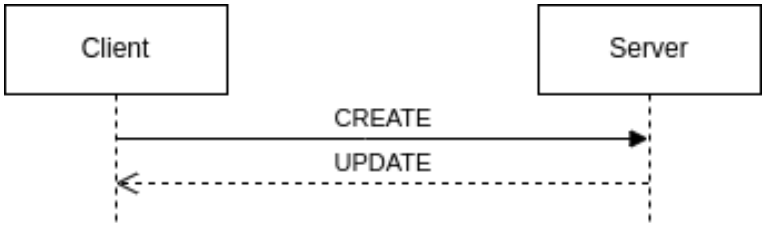

Figura 2: Scenario di creazione della partita

2.3 Unione/abbandono di una partita

Se un giocatore volesse unirsi ad una lobby già esistente, il client manderà al server un messaggio di tipo `JOIN`. Il server risponderà in broadcast a tutti i client connessi alla stessa lobby con un messaggio di tipo `UPDATE` contenente la nuova lista di giocatori e la nuova plancia del giocatore appena unito.

Analogamente, nel caso in cui un giocatore volesse abbandonare la lobby, il client manderà un messaggio di `LEAVE` e il server risponderà in broadcast con un messaggio di `UPDATE` contenente la lista di giocatori aggiornata.

Nel caso di un eventuale crash del server, i client potranno riconnettersi alla loro vecchia partita mandando dei messaggi di `JOIN` contenenti l'id della vecchia partita e lo stesso username. Dopo che tutti i giocatori si saranno uniti, la partita riprenderà.

Nota: la fase di *lobby* di una partita è l'unico momento in cui un giocatore ha la possibilità di unirsi e abbandonare la partita. Se un giocatore si disconnette a partita già iniziata, la partita verrà terminata.

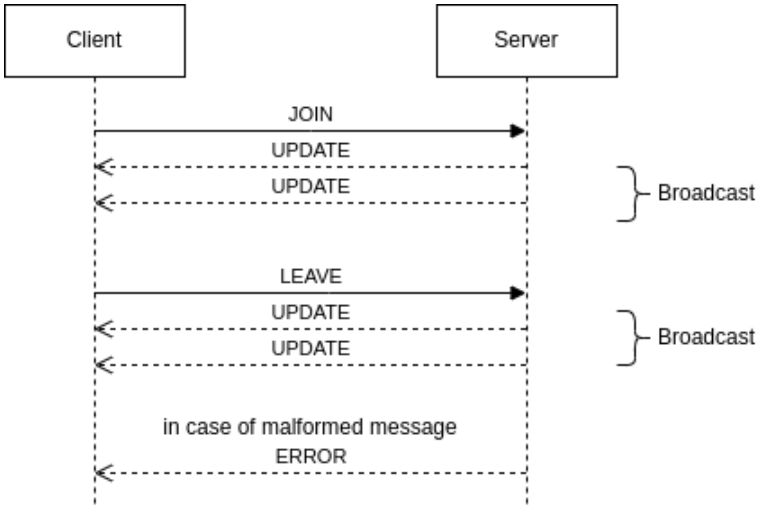

Figura 3: Scenario di unione ad una partita esistente

2.4 Fase di gioco

La fase di gioco inizia non appena si uniscono tutti i giocatori. I client manderanno dei comandi corrispondenti alle possibili azioni al server, il quale risponderà con `UPDATE` in broadcast nel caso la modifica vada a buon fine o con un `ERROR` diretto al mittente in caso di azione non permessa o messaggio malformato.

Nel caso un comando causi la fine della partita, un messaggio di `END` verrà mandato in broadcast contente la lista dei vincitori della partita.

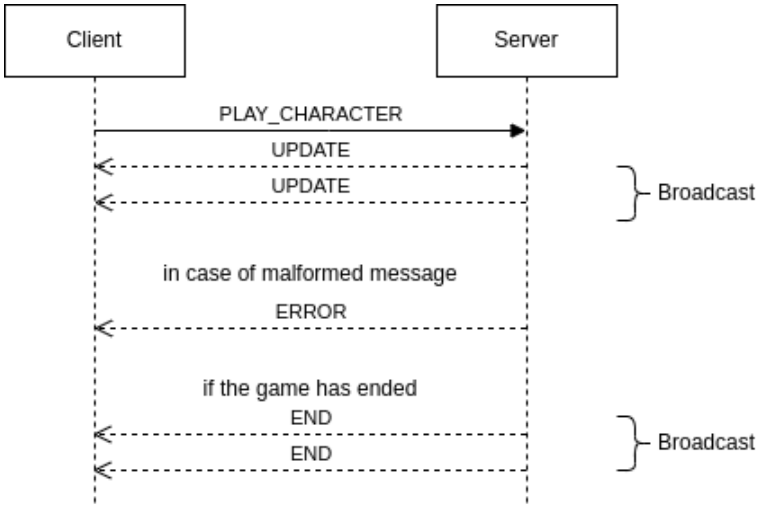

Figura 4: Scenario di gioco, in questo esempio con un comando per giocare un personaggio

2.5 Ping

Durante tutta la connessione ad una partita, il server manderà un messaggio di `PING` ai vari client ad intervalli regolari. A questi messaggi i client risponderanno con il corrispondente `PONG`.

Se un client non risponde in tempo al ping, il server terminerà la partita di cui fa parte e manderà un messaggio di `END` in broadcast agli altri membri della partita.

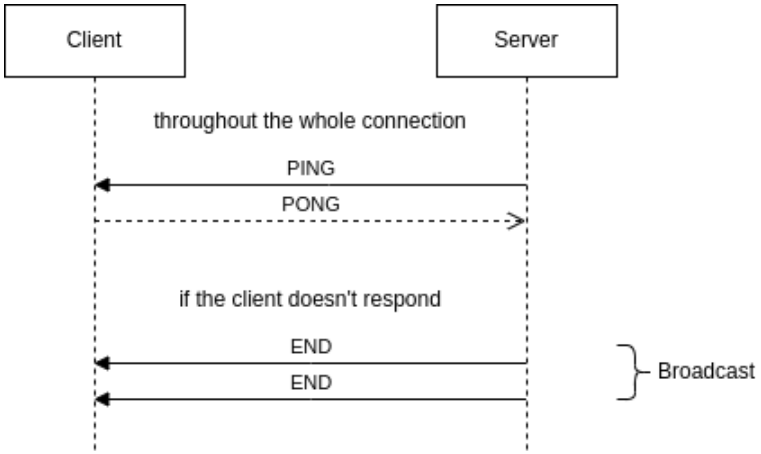

Figura 5: Ping tra server e client

2.6 Esempio di partita

In figura 6 verrà riportato uno scambio di esempio tra un client ed il server. Un giocatore, dopo aver scelto una partita dalla lista di partite disponibili vi si unisce. In seguito prova ad usare l'effetto di una carta personaggio in un momento non consentito; la mossa successiva, però, è legittima. Infine il server conclude la partita a causa della vittoria di un giocatore. Per chiarezza del diagramma i messaggi di ping sono stati omessi.

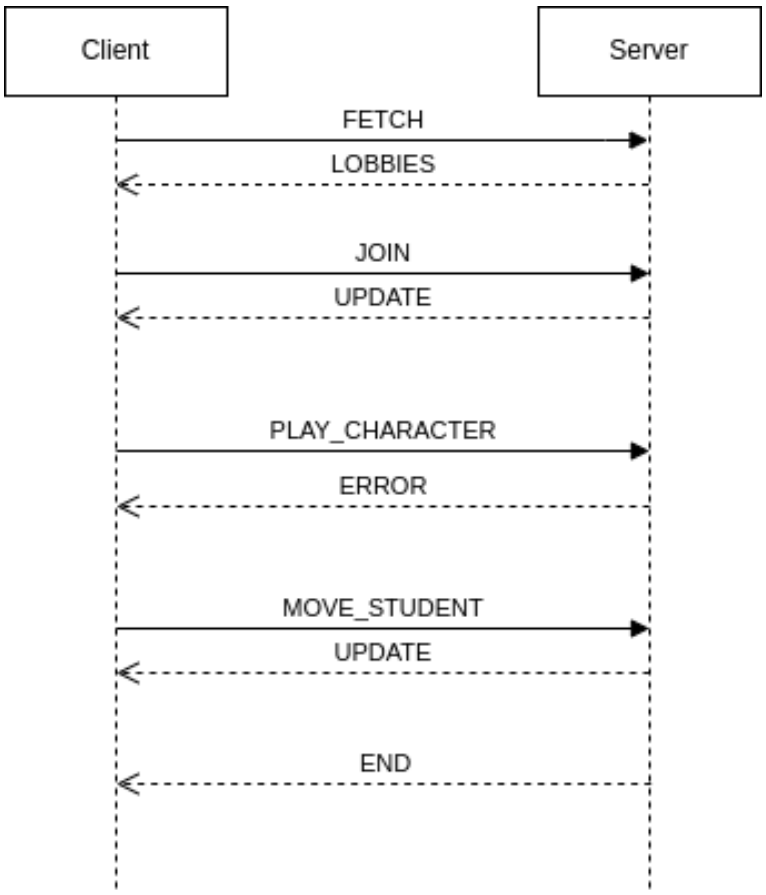

Figura 6: Scambio d'esempio

3 Dettaglio messaggi client

In questa sezione verranno descritti tramite degli esempi i vari messaggi inviati dai client al server

3.1 FETCH

Richiede una lista di partite libere al server.

```
{
  "type": "FETCH"
}
```

3.2 CREATE

Richiede la creazione di una nuova lobby con i parametri indicati. Il giocatore che crea la lobby vi è automaticamente aggiunto.

```
{
  "username": "ann",
  "type": "CREATE",
  "arguments": [
    {
      "nPlayers": 2
      "expert": true
```

```
    }
  ]
}
```

3.3 JOIN e LEAVE

Rispettivamente richiedono l'aggiunta o la rimozione del giocatore con lo username specificato all partita con l'identificativo dato.

```
{
  "gameId": 1234567890,
  "username": "ann",
  "type": "JOIN"
}

{
  "gameId": 1234567890,
  "username": "ann",
  "type": "LEAVE"
}
```

3.4 Comandi di gioco

Questa serie di comandi rappresentano le varie mosse eseguibili da un giocatore durante una partita, come ad esempio il movimento degli studenti o di madre natura. Ogni comando possiede l'identificativo della partita a cui appartiene, lo username del giocatore che lo esegue, il tipo di comando e una lista di parametri variabile in base al tipo di comando.

Per scopi documentativi, sono stati aggiunti dei commenti delimitati da //.

```
{
  "gameId": 1234567890,
  "username": "ann",
  "type": "CHOOSE_MAGE",
  "arguments": ["WIZARD"] // valori dall'enum Mage
}

{
  "gameId": 1234567890,
  "username": "ann",
  "type": "PLAY_ASSISTANTS",
  "arguments": ["CAT"] // valori dall'enum AssistantType
}

{
  "gameId": 1234567890,
  "username": "ann",
  "type": "MOVE_STUDENT",
  // Solo uno dei seguenti oggetti sarà mai mandato, sono stati riportati
  // entrambi per scopi documentativi
  "arguments": [
    {
```

```
      "source": "HALL",
      "color": "RED" // valore dall'enum PieceColor
    },
    {
      "source": "ISLAND",
      "color": "RED", // valore dall'enum PieceColor
      "destination": 0
    }
  ]
}

{
  "gameId": 1234567890,
  "username": "ann",
  "type": "PLAY_CHARACTER",
  "arguments": [
    {
      "character": "PRIEST", // Valore dall'enum CharacterType
      "steps": [
        {
          "key1": "value1",
          "key2": "value2"
        },
        {
          "key1": "value1",
          "key2": "value2"
        }
      ]
    }
  ]
}

{
  "gameId": 1234567890,
  "username": "ann",
  "type": "MOVE_MN",
  "arguments": [3] // Numero di isole di cui si vuole muovere madre natura
}

{
  "gameId": 1234567890,
  "username": "ann",
  "type": "PICK_CLOUD",
  "arguments": [2] // Id della nuvola che si vuole prendere
}
```

3.5 PONG

```
{
  "id": 1234567890
```

```
  "type": "PONG"
}
```

4 Dettaglio messaggi server

In questa sezione verranno descritti tramite degli esempi i vari messaggi inviati dal server ai client.

4.1 LOBBIES

Questa risposta è mandata in seguito ad un comando di `FETCH`. Essa contiene una lista di tutte le lobby che non sono ancora piene, con le relative caratteristiche.

```
{
  "type": "LOBBIES"
  "lobbies": [
    {
      "id": 1234567890,
      "nPlayers": 2,
      "expert": true
    }
  ]
}
```

4.2 UPDATE

Risposta di successo mandata in seguito ad un comando che modifica lo stato del gioco. Contiene tutti e soli gli oggetti modificati dal comando a cui è risposta. Nell'esempio riportato in seguito è stata riportata la massima estensione del comando.

```
{
  "id": 1234567890,
  "type": "UPDATE",
  "update": {
    "phase": "PhaseName",
    "currentPlayer": "ann",
    "playerList": ["ann", "bob"],
    "professors": [
      {
        "color": "RED",
        "owner": "bob"
      }
    ],
    "boards": [
      {
        "username": "ann",
        "entrance": ["RED", "BLUE"],
        "assistants": ["CAT"],
        "lastPlayedAssistant": "ELEPHANT",
        "hall": ["RED", "BLUE"],
```

```
        "towers": ["BLACK", "BLACK"],
        "coins": 5
      }
    ],
    "islandList": [[0], [1,2,3], [4,5,6], [7,8,9,10,11]],
    "islands": [
      {
        "ids": [1,2,3],
        "students": ["RED", "BLUE"],
        "towers": ["BLACK", "BLACK", "BLACK"],
        "blocks": 2
      }
    ],
    "motherNature": {
      "position": 10
    },
    "usedCharacter": false,
    "characters": [
      {
        "type": "PRIEST",
        "students": ["RED", "BLUE"],
        "price": 3,
      }
    ],
    "isSackEmpty": false,
    "clouds": [
      {
        "id": 1,
        "students": []
      }
    ]
  }
}
```

4.3 END

La risposta di tipo `END` viene mandata al posto di un `UPDATE` se il comando ricevuto ha causato la fine del gioco. La risposta di `END` viene anche mandata per segnalare la terminazione del gioco in caso di disconnessione di un giocatore.

```
{
  "id": 1234567890,
  "type": "END",
  "winners": ["ann"]
  "reason": "A winner has been found"
}

{
  "id": 1234567890,
  "type": "END",
  "winners": []
```

```
  "reason": "A player was disconnected"
}
```

4.4 ERROR

Mandato in risposta ad un comando malformato o non permesso in quel momento della partita in corso.

```
{
  "id": 1234567890,
  "type": "ERROR",
  "reason": "This action is not allowed now"
}
```

4.5 PING

```
{
  "id": 1234567890,
  "type": "PING"
}
```