# DOCUMENTAZIONE PROGETTO NETWORKING SU JAVA: BATTAGLIA NAVALE

**Link repository:**https://github.com/UrFavWallCrawler/Battleship/

## Descrizione dell'applicazione

Il nostro progetto consiste in un'implementazione online del gioco della Battaglia Navale secondo un paradigma Client-Server. Un client, dopo essersi connesso, attende fino all'arrivo di un altro client in stato di attesa (che si è appena connesso o che ha terminato una partita); a questo punto il server crea una partita tra i due client, che si divide in una fase iniziale dove i giocatori posizionano le navi (che non si svolge a turni) e, quando entrambi sono pronti, la fase del gioco vero e proprio, dove a turno i giocatori tentano di affondare le navi dell'avversario.

La logica del gioco (determinare se un colpo sia andato a segno o abbia preso acqua, quando le navi vengono affondate, se tutte le navi di un giocatore siano state affondate, ecc.) viene gestita lato server (eccetto per il controllo della validità degli input, gestito dai client); le applicazioni client agiscono esclusivamente come interfacce con l'utente, inviando input al server e visualizzando le risposte fornite da esso, aggiornando il campo da gioco che tiene traccia dei colpi sparati all'avversario con il loro esito (acqua, colpito, affondato), quello con le navi del giocatore con l'esito dei colpi sparati dall'avversario, e informando il giocatore dell'esito della partita al suo termine. Inoltre, in qualsiasi momento, entrambi i giocatori hanno l'opzione di abbandonare la partita (senza però disconnettersi dal server), facendo vincere immediatamente l'avversario, e di inviare messaggi che verranno mostrati all'avversario in una semplice chat.

## Descrizione del protocollo

Il nostro protocollo si basa sullo scambio di messaggi testuali in formato JSON; di base tutti i pacchetti hanno la forma:

```
{
     "message": <STRINGA che identifica il tipo di pacchetto>
     "content": <OGGETTO che contiene informazioni specifiche al
pacchetto; può anche avere valore null>
}
```

Il protocollo è fortemente stateful, dato che l'interpretazione dei pacchetti dipende quasi sempre da quelli ricevuti in precedenza, non solo per il server (che si deve basare sullo stato attuale della partita) ma anche per il client.

La dimensione del campo di gioco ed il numero delle navi sono noti a priori: Il campo è composto da una matrice di 10x10 celle (che internamente sono indicizzate con due valori separati che vanno da 0 a 9), e ogni giocatore dispone di 8 navi: 1 nave lunga 5 celle, 2 navi lunghe 4 celle, 2 navi lunghe 3 celle, e 3 navi lunghe 2 celle;  il numero delle navi è noto a priori, ma le celle relative ad ogni nave sono inviate con il pacchetto del client (che consentirebbe eventualmente di avere navi non rettilinee, con forme ad "L" o a "T").

Adesso andremo a definire i pacchetti inviati da server e client, definendone la struttura JSON e la funzione all'interno dell'applicazione, per poi creare un diagramma di flusso che descrive la struttura del protocollo (vista dal lato del client).

## Pacchetti inviati dal server

*WAIT*

Primo pacchetto ad essere inviato al client; segnala al client che la connessione è avvenuta con successo e che deve attendere l'inizio di una partita (segnalato da un pacchetto GAME).

Struttura JSON:

```
{
"message": "WAIT",
"content": null
}
```

*GAME*

Segnala ai client (i due giocatori coinvolti in una partita) che è stata avviata la partita, e che ha avuto inizio la fase di posizionamento delle navi; il server sa a priori quante navi deve posizionare ciascun giocatore, e quando tutte le navi sono state posizionate invierà ad entrambi un pacchetto START

Struttura JSON:

```
{
"message": "GAME",
"content": null
}
```

*START*

Segnala ai client che è terminata la fase di piazzamento delle navi ed è iniziata la fase dove i giocatori si spareranno dei colpi a turno; Quando riceve questo messaggio, *il client presume che non sia il suo turno*, e attende un pacchetto REPLY che descrive il colpo in arrivo dall'avversario o un pacchetto TURN che indica che è il suo turno.

Struttura JSON:

```
{
"message": "START",
"content": null
}
```

*TURN*

Segnala al client che è il suo turno, e che quindi può inviare un pacchetto SHOT; il turno del client finisce all'arrivo di un pacchetto REPLY, che descrive l'esito del colpo sparato verso l'altro giocatore.

Struttura JSON:

```
{
"message": "TURN",
```

```
"content": null
}
```

*REPLY*

Questo pacchetto contiene l'esito di un colpo, che può assumere tre stadi diversi (hit, miss, sink), e la cella (o celle) a cui si riferisce; questo pacchetto viene inviato ad *entrambi* i giocatori; il giocatore che ha precedentemente inviato un pacchetto SHOOT sa che il REPLY si riferisce ad esso (contiene l'esito del suo colpo), e che questo determina la fine del suo turno; mentre il giocatore del quale non è il turno (che ha precedentemente ricevuto un altro pacchetto REPLY, che quindi ha segnato la fine del suo turno) sa che questo pacchetto si riferisce al colpo inviato *dall'avversario*.

Struttura JSON:

```
{
"message": "REPLY",
"content":  {
     "result": <"HIT"/"MISS"/"SINK">
     "cells":[
          {
               "xCoord":<numero coordinata x cella>,
               "yCoord":<numero coordinata y cella>
          },
          <se result è "SINK" cells conterrà più oggetti,
          che rappresta la nave affondata>
          ]
     }

}
```

*END*

Questo pacchetto comunica ai client che la partita è terminata (perché tutte le navi di un giocatore sono state affondate oppure perché uno dei due giocatori si è arreso tramite l'invio di un pacchetto FLAG); il pacchetto indica l'esito della partita, indicando se il destinatario ha vinto o ha perso. Questo pacchetto può essere inviato in qualsiasi momento dopo l'invio del pacchetto GAME.

Struttura JSON:

```
{
"message": "END",
"content":  {
     "result": <"WIN"/"LOSE">
     }
}
```

*SAY*

Questo messaggio invia il messaggio inviato al server da un giocatore all'avversario, in modo che questo possa essere visualizzato. Questo pacchetto può essere inviato in qualsiasi momento dopo l'invio del pacchetto GAME

Struttura JSON:

```
{
"message": "SAY”,
"content":  {
     “said”: <stringa con messaggio chat del giocatore>
}
```

## Pacchetti inviati dal client

### *SHIP*

Questo pacchetto viene inviato dal client dopo aver ricevuto un pachetto GAME; il client invierà uno di questi pacchetti per ogni nave che intende piazzare (verranno quindi inviati 5 di questi pacchetti).

Struttura JSON:

```
{
"message": "SHIP”,
"content":  {
     “id”: <numero che identifica la nave>,
     “cells”:[
     {
          “xCoord”:<numero coordinata x cella>,
          “yCoord”:<numero coordinata y cella>
     }, <l’array avrà tanti elementi quante sono le celle da cui è
composta la nave>
            ]
     }


}
```

### *SHOOT*

Questo pacchetto viene inviato durante il turno di un giocatore (segue quindi un pacchetto TURN) ed viene usato dal giocatore per sparare ad una cella della griglia dell’avversario: l’esito dello sparo verrà indicato in un successivo pacchetto REPLY.

Struttura JSON:

```
{
"message": "SHIP”,
"content":  {
     “cells”: {
          “xCoord”:<numero coordinata x cella>,
          “yCoord”:<numero coordinata y cella>
     }
}
```

*FLAG*

Questo pacchetto può essere inviato in qualsiasi momento dopo l'avvio della partita (dopo l'arrivo di un pacchetto GAME) e consente al client di arrendersi; è seguito da un pacchetto END da parte del server (il player che si è arreso perderà e il suo avversario vincerà);

Struttura JSON:

```
{
"message": “FLAG",
"content": null
}
```

*SAY*

Questo pacchetto può essere inviato in qualsiasi momento dopo l'avvio della partita (arrivo di un pacchetto GAME) e consente al client di inviare un messaggio testuale all'avversario.

Struttura JSON:

```
{
"message": "SAY”,
"content":  {
     “said”: <stringa con messaggio chat del giocatore>
}
```