

Battleship

 Course	 <u>Distributed Programming</u>
 Date	@October 13, 2023
 Type	Exam
 Status	Completed

Descrizione

L'applicazione **Battleship** presenta una struttura di tipo *client-server*, dove il ruolo dei client viene svolto dai player del gioco.

Requisiti del server

Il *server* deve:

[FS1] autenticare i player e rifiutare la connessione se esiste già un player con quell'username,

[FS2] creare un tavolo ogni due utenti che si collegano,

[FS3] permettere agli utenti di posizionare le navi,

[FS4] permettere agli utenti di inviare mosse per colpire le navi nemiche,

[FS5] comunicare agli utenti se hanno mancato o meno le navi nemiche,

[FS6] aggiornare le game boards dopo ogni turno di gioco,

[FS7] comunicare la vittoria alle due parti,

[FS8] gestire disconnessioni brusche dei client.

Requisiti del client

Il *client* deve:

[FC1] potersi connettere al server usando il proprio indirizzo IP e numero di porta,

[FC2] poter inserire un username,

[FC3] poter posizionare le proprie navi,

[FC4] poter eseguire delle mosse per poter affondare le navi nemiche,

[FC5] poter visualizzare i messaggi ricevuti dagli altri utenti,

[FC6] poter sapere se ha colpito o mancato le navi nemiche,

[FC7] poter sapere se ha vinto o a perso,

[FC8] potersi disconnettere.

Struttura

Si è deciso di fornire un’interfaccia grafica basilare per i player. A tale scopo si è adottata l’architettura MVC (Model View Controller), usando SceneBuilder per ottenere il file Document in formato FXML che descrive l’interfaccia, mentre il Controller è stato ottenuto per generazione automatica dell’IDE Eclipse stesso, a partire dal Document. Per rappresentare i messaggi scambiati tra i client e il server e viceversa, è stata predisposta l’apposita classe `Message`.

La rappresentazione della tavola di gioco è riportata nella classe `Table`, le tavole di gioco sono rappresentate dalla classe `GameBoard` e la classe `Player` rappresenta invece il giocatore.

Server

La classe `Server` contiene la logica del server. Il server mantiene la corrispondenza tra l’username del giocatore ed il giocatore nella struttura `users` e la corrispondenza tra il giocatore ed il tavolo di gioco a cui appartiene nella struttura `tables`. Il server, quando viene creato nel metodo main, crea la propria socket. Inoltre, è necessario per i thread creati sapere in quale momento di gioco si trovano i player in modo da poter inviare i messaggi opportuni e ciò accade per mezzo dell’attributo `todo`. Questi quattro attributi sono statici e sono quindi comuni a tutti i thread della classe Server.

Gli altri attributi sono propri di ogni singola istanza distinta del Server e mantengono delle informazioni che dipendono dal contesto, come un utente che è in attesa di qualcun altro per creare un tavolo di gioco, ecc.

Quando il server è in esecuzione, esegue in un loop infinito il metodo `acceptMessage()` che serve a stabilire di cosa si deve occupare il server sulla base del contenuto del messaggio ricevuto. Il protocollo adottato prevede che la prima parola della stringa ricevuta sia l’azione da gestire e sulla base di questa viene attivato un *handler* distinto:

- `username`, attiva il metodo `handleConnection` che verifica la conformità dell’username inserito ed eventualmente autentica l’utente, dopodiché crea un thread che si occupa di inviargli un messaggio con un feedback sull’esito dell’operazione,

- `placement`, attiva il metodo `handlePlacement` che posiziona la nave nelle coordinate specificate dall'utente, dopodiché crea un thread che si occupa di inviargli un messaggio con un feedback sull'esito dell'operazione,
- `play`, attiva il metodo `handleMove` che effettua un attacco alla game board dell'avversario sulla base delle coordinate specificate dall'utente, dopodiché crea un thread che si occupa di inviargli un messaggio con un feedback sull'esito dell'operazione,
- `disconnect`, attiva il metodo `handleDisconnection` che gestisce la disconnessione dell'utente.

I thread che vengono creati hanno la sola funzione di inviare messaggi, mentre quello principale li riceve e li processa.

Client

La classe `Client` mantiene la logica del client. Esso ha una serie di attributi statici, che sono comuni a tutte le istanze del thread che servono a tenere conto delle informazioni di contesto, la propria socket e le informazioni ad essa correlate, come il numero di porta e l'indirizzo IP.

Abbiamo due costruttori:

- uno vuoto che viene invocato per creare il thread,
- uno che prende per parametri delle *BooleanProperties* che viene usato nel main per creare il client, facendo sì che quando la logica che esso contiene cambia i valori di questi parametri, il cambiamento si rifletta sugli elementi dell'interfaccia.

Per effettuare la connessione al server, il client usa il metodo `connect`, che si occupa di creare una socket, inviare una richiesta di connessione per un determinato username fornito in input ed aspettare una risposta.

Per effettuare una disconnessione dal server, il client usa il metodo `disconnect`, che si occupa di inviare una richiesta di disconnessione e di chiudere la socket.

Il client invoca il metodo `placeShip` per posizionare una nave ed il metodo `sendMove` per inviare al server la mossa che desidera fare, dopodiché ne processa il feedback.

Il metodo `receiveMove` viene invece invocato dal Task avviato dalla GUI per ricevere i messaggi dal server quando l'utente è in attesa delle mosse dell'avversario.

Table

La classe `Table` rappresenta il tavolo di gioco e mantiene i due *Player* che ne fanno parte ed una mappa con la corrispondenza tra il *Player* e la sua *GameBoard*. Oltre i getter, ha un

metodo utilitario che dato un player, restituisce l'avversario.

GameBoard

La classe `GameBoard` rappresenta la board di gioco di un *Player* e contiene una matrice con un numero predefinito di righe e di colonne, popolata inizialmente con il carattere ‘-’, una mappa che fa corrispondere ad ogni tipologia di nave la sua lunghezza e degli attributi per mantenere dati di contesto.

Tale classe consente di invocare il metodo `placeShip` per posizionare una nave ed il metodo `makeMove` per effettuare una mossa. Quest’ultimo restituisce una stringa che consente al chiamante di inviare un feedback testuale all’utente.

Quando viene posizionata una nave, le corrispondenti celle della matrice presentano la lettera ‘S’. Quando una cella di una nave viene colpita, la lettera ‘S’ viene sostituita con la lettera ‘X’. Quando viene colpita una cella che non è una nave, viene inserito il carattere ‘0’.

Player

Un `Player` è descritto agli occhi del server come una terna di attributi: username, numero di porta e indirizzo IP.

Message

Un `Message` contiene del testo, il numero di porta e l’indirizzo IP dell’utente che lo invia.

Communicator

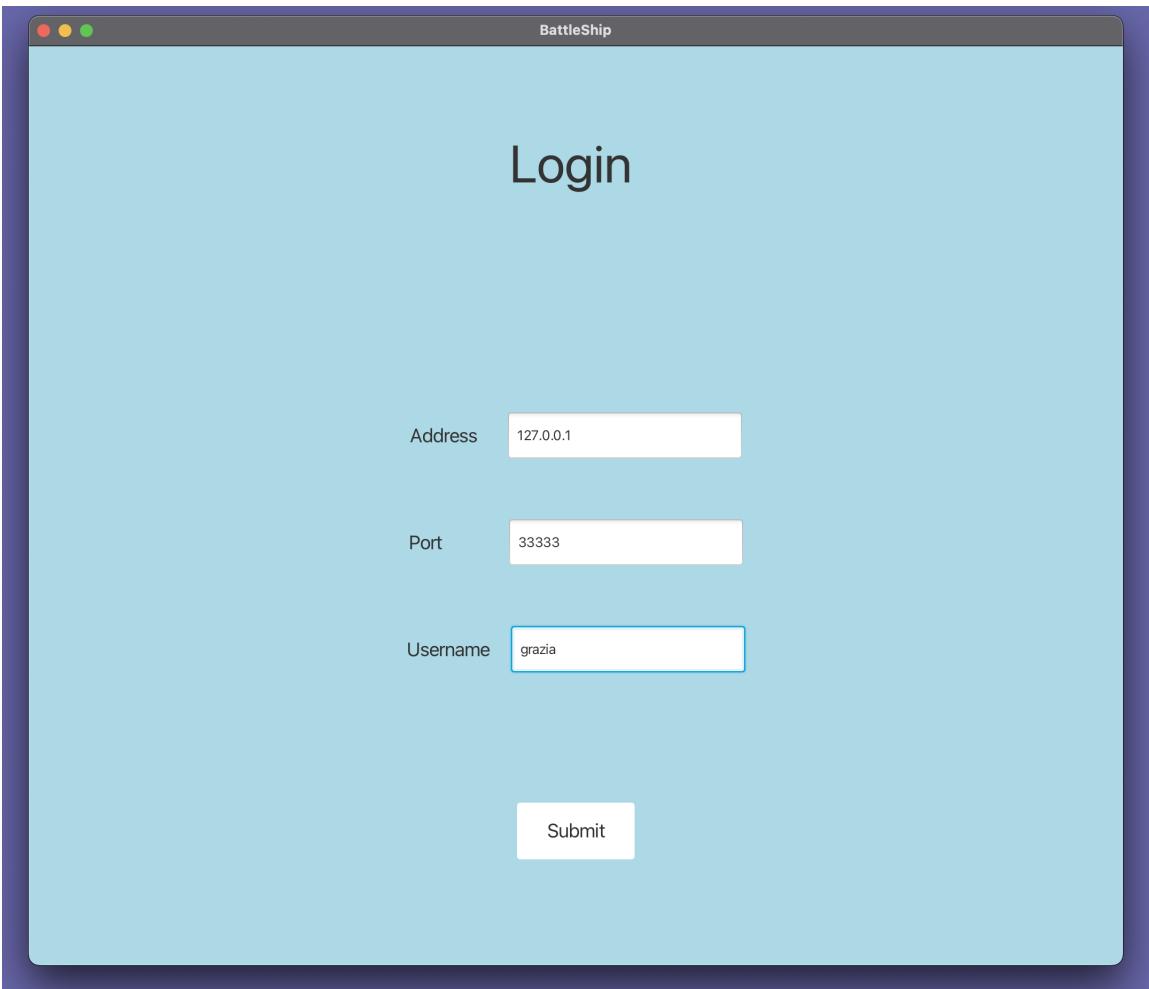
Sia il *Client*, che il *Server* estendono la classe estratta `Communicator`, poiché entrambi devono inviare e ricevere messaggi usando le *DatagramSocket* UDP.

I due metodi messi a disposizione sono:

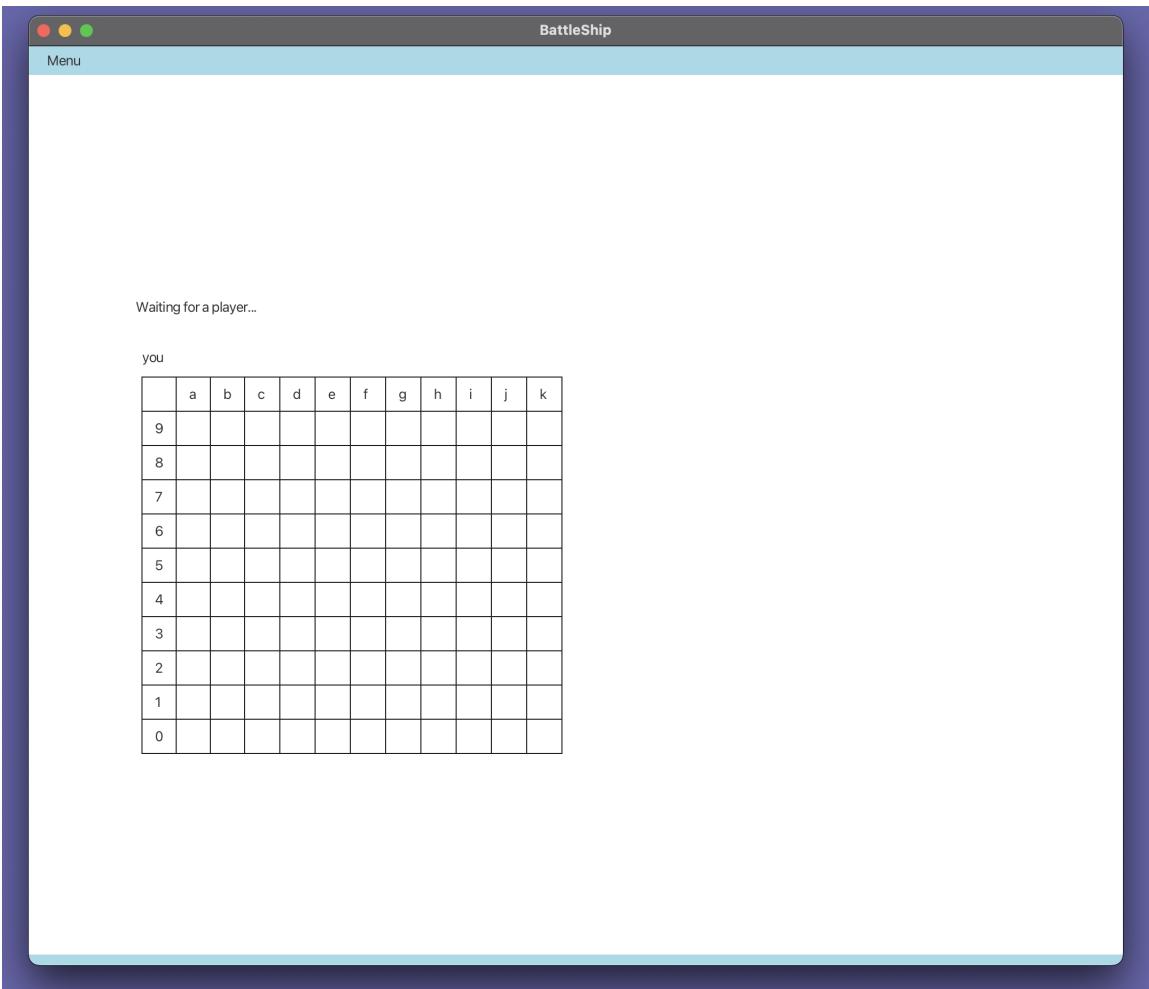
- `void sendMessage(DatagramSocket sock, String message, InetAddress IPAddress, int port)`, che consente di inviare un messaggio,
- `Message receiveMessage(DatagramSocket sock)`, che consente di ricevere un messaggio.

Funzionamento

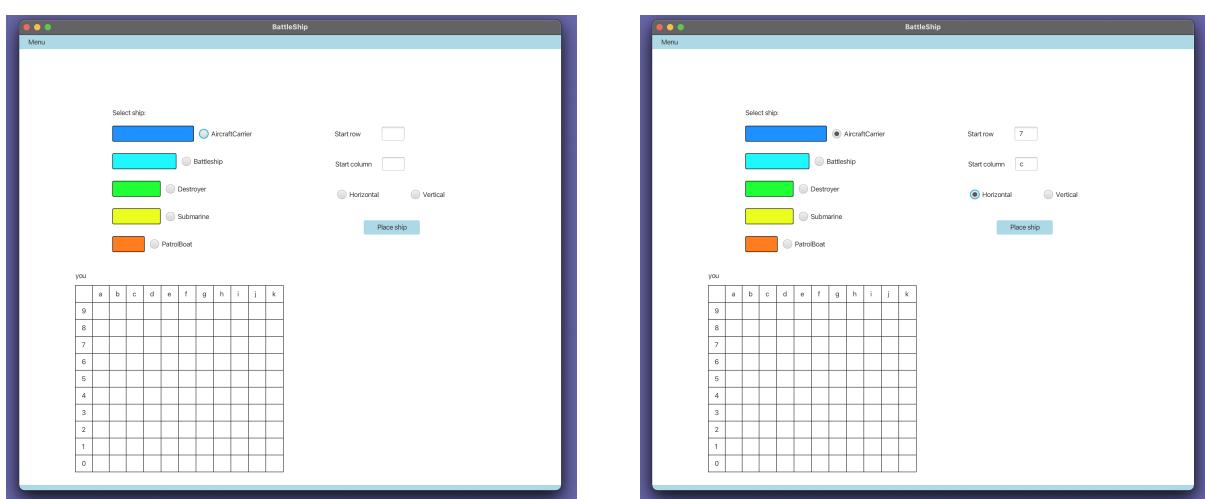
L’utente può effettuare il login al gioco inserendo indirizzo IP, il numero di porta (nello specifico 33333) e lo username. Gli viene dato un messaggio di errore in caso di campi non validi o username già in uso.



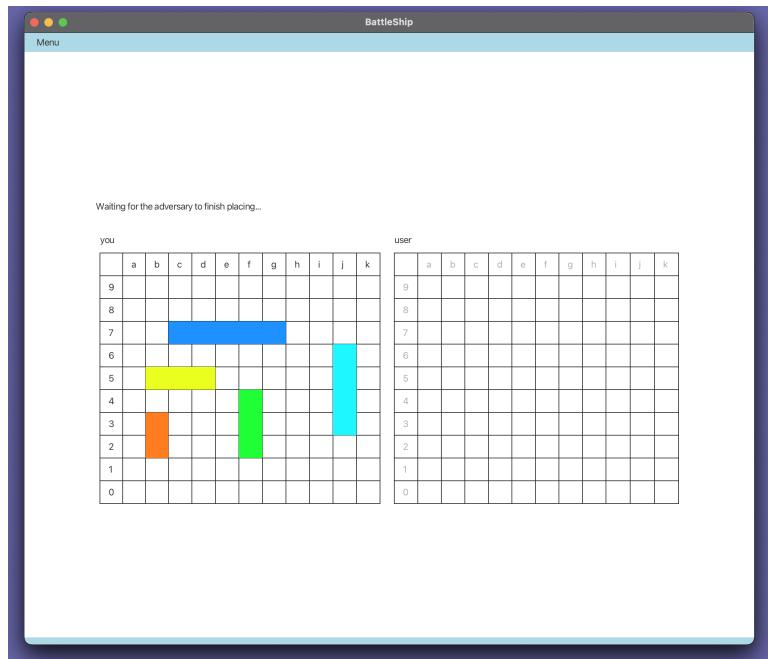
Il primo player resta in attesa che un secondo si colleghi al gioco, in modo che possa essere formato un tavolo.



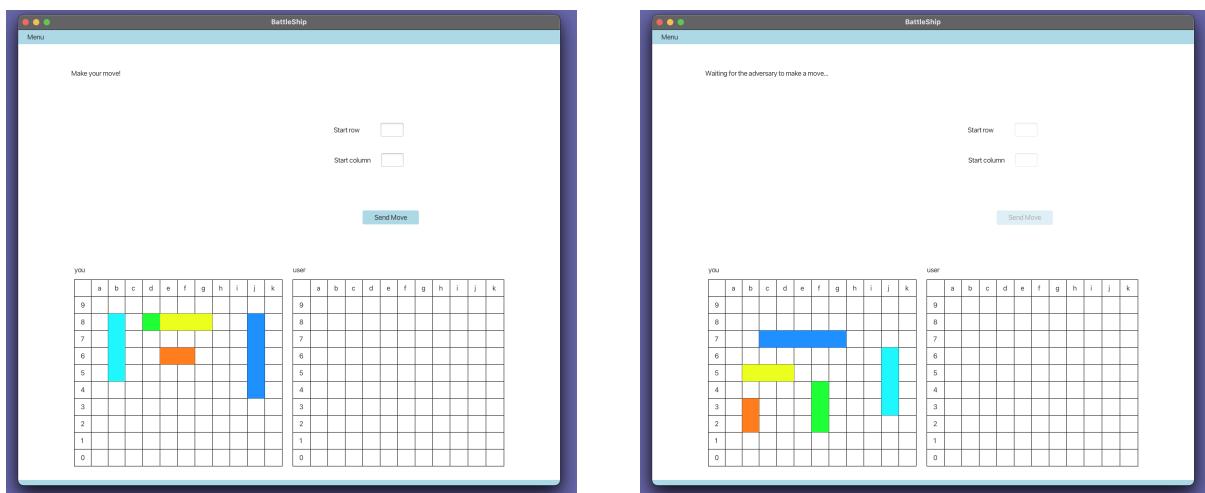
Una volta formatosi il tavolo, i due utenti hanno la possibilità di posizionare le navi. Alcuni controlli sui campi inseriti sono presenti.



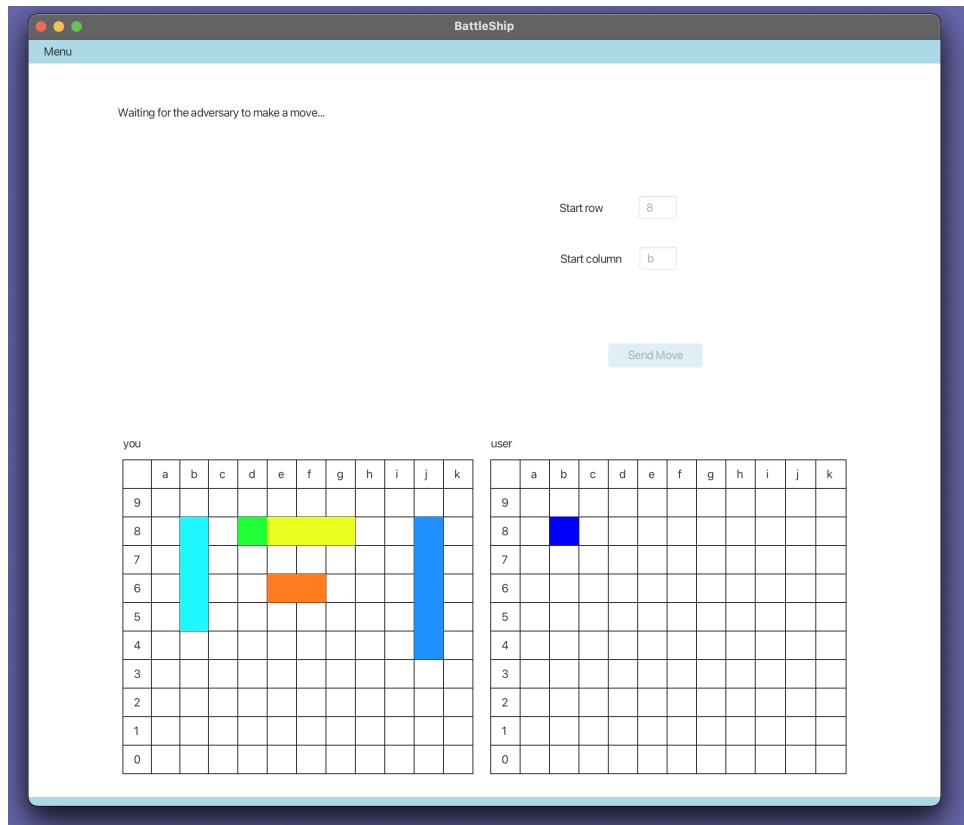
Il primo che finisce di posizionare le navi attende che anche il secondo termini.



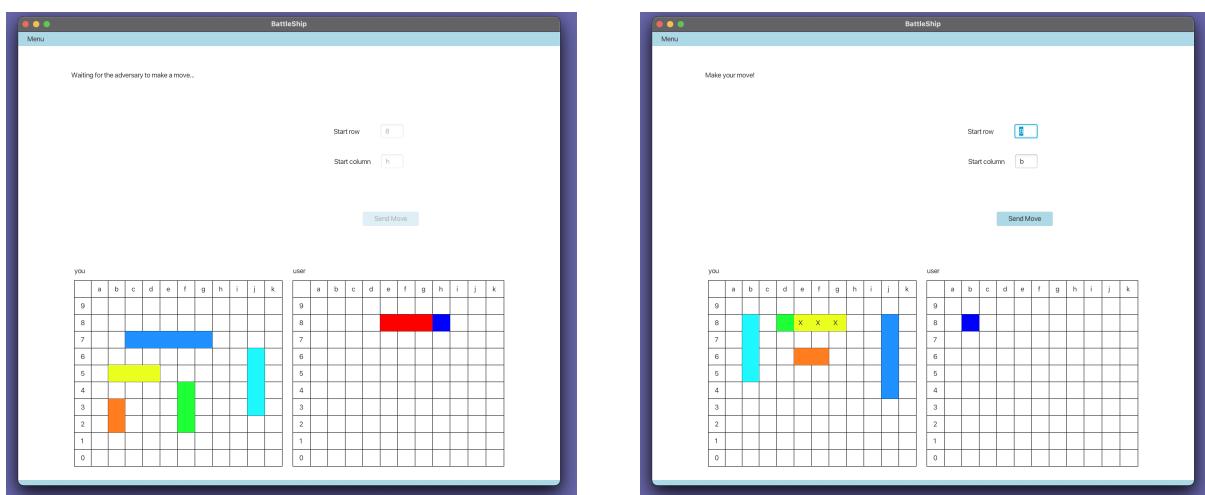
A turno i due player hanno la possibilità di colpire le navi dell’altro. Se una nave viene colpita, la corrispondente cella si colora di rosso nella tabella dell’utente ed appare una X nella tabella dell’avversario. Se nessuna nave viene colpita, la cella si colora di blu.



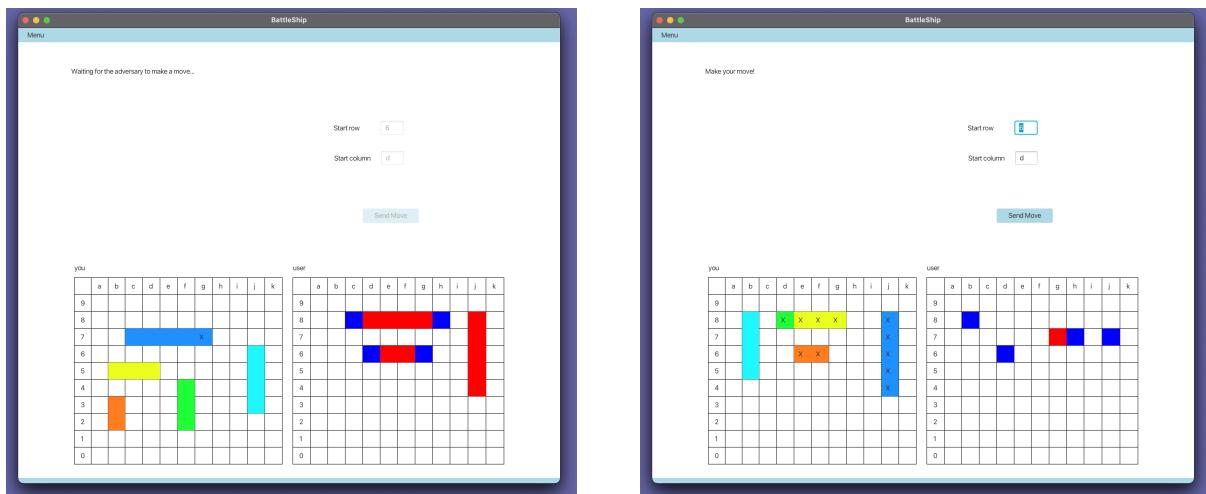
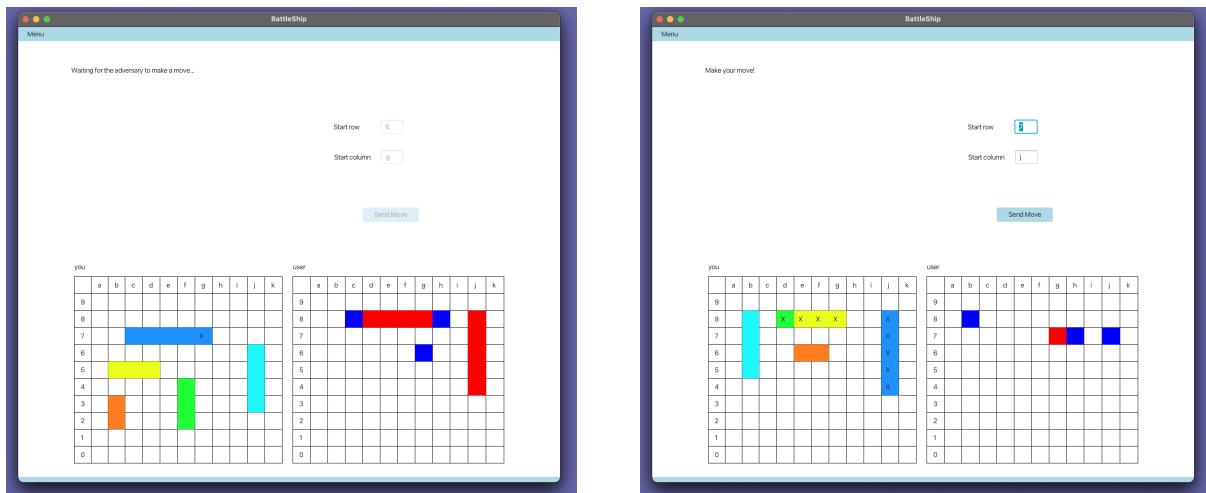
Ad esempio nella seguente immagine, l’utente ha effettuato una *miss*, quindi tocca all’avversario.



Nelle seguenti immagini invece vediamo il caso dell’utente che ha effettuato the *hit* consecutive (in rosso) e solo dopo che ha effettuato una *miss* (in blu), è il turno dell’avversario.



La partita continua in questo modo fino a che tutte le celle dove erano presenti le navi di uno dei due non sono state tutte colpite.



Quando uno dei due vince, riceve la notifica di vittoria, mentre l'avversario riceve quella di sconfitta. Poi entrambi tornano alla schermata di login.

