Nel file **client.js** vengono implementate tutte le funzionalità lato client.

Per prima cosa viene effettuata la registrazione al backend, dichiarando la variabile *socket = io().* La connessione verrà notificata dall'evento *connect.*

Alla ricezione di un evento *'join',* preleva la pagina il suo username dalla pagina html di gioco ed invia un evento contenente la sua username al server, che terrà traccia delle sue informazioni relative alla corrente sessione di gioco. Viene inoltre riprodotta una musica di sottofondo nella sessione stessa.

Algoritmo 1 Ricezione evento 'join'

```
socket.on('join', function(gameId) {
  var username = $('#myUsername').text();
  console.log('Username inviato ' + username);
  socket.emit('new player', username);
  Game.initGame();
      $('#messages').empty();
  $('#disconnected').hide();
  $('#waiting-room').hide();
  $('#game').show();
  $('#game-number').html(gameId);
  $('#myScore').html("0");
  $('#oppScore').html("0");
  socket.emit('oppNotification');    //Notify opponent's username
  myMusic.play();
      }
```

Spesso capita che il client invii un username nullo al server, probabilmente perché la pagina html del gioco, da cui egli prende il suo username, non si è ancora caricata del tutto. Allora il server riceve un evento dal server dove viene richiesto di rinviare la sua username: tale meccanismo può essere visto come una sorta di loop da cui si esce non appena il client sarà riuscito ad inviare un username diverso da null.

Algoritmo 2 Ricezione evento 'errorUsername'

```
socket.on('errorUsername',function(){
  socket.emit('new player',$('#myUsername').text());
});
```

E' stata creata una funzione **sound** che permette la creazione di un elemento audio a partire da path che specifica un file musicale, e i cui metodi *play()* e *stop()* permettono di riprodurre o stoppare la riproduzione del file audio specificato.

Algoritmo 3 Funzione sound()

```
function sound(src) {
  this.sound = document.createElement("audio");
  this.sound.src = src;
  this.sound.setAttribute("preload", "auto");
  this.sound.setAttribute("controls", "none");
  this.sound.style.display = "none";
  document.body.appendChild(this.sound);
  this.play = function() {
    this.sound.play();
  }
  this.stop = function() {
    this.sound.pause();
  }
}
```

Il client in prevalenza riceve eventi dal lato server e, in conseguenza di ciò, avviene qualcosa:

- alla ricezione dell'evento *'update'* viene aggiornato lo stato del gioco del client;
- alla ricezione degli eventi *'scoreMe'* e *'scoreOPP'* aggiorna rispettivamente lo score del giocatore e del suo avversario nella pagina html di gioco;
- alla ricezione degli eventi *'gameover_winner'* o *'gameover_loser'* viene decretata la fine del partite corrente ricevendo una notifica testuale ed audio diversa che dipende da se il giocatore ha vinto o perso la partita corrente;
- alla ricezione degli eventi di *'valid_shot'* e *'invalid_shot'*, che indicano un colpo valido o invalido (a seconda se il client ha precedentemente sparato su una casella libera o meno) vengono riprodotto rispettivamente il suono di un cannone da guerra o una notifica di warning;
- alla ricezione degli eventi di *'hit_ship'* e *'miss_ship'*, che vengono ricevuti a secondo del fatto che il colpo precedentemente inviato dal giocatore abbia colpito o meno una nave avversaria. Se una nave è stata colpita, allora viene riprodotto il suono di un esplisone, altrimenti viene riprodotto il tipico suono di *'splash'*;

Gli eventi che vengono inviati dal client verso il server sono:

- *'shot':* viene inviato quando viene chiamata la funzione *sendShot()*, cioè quando l'utente clicca su una casella della griglia avversaria durante il proprio turno di gioco. Come abbiamo visto nel file **gameServer.js**, alla ricezione di tale evento vengono effettuate varie verifiche (colpo valido / invalido, nave colpito / non colpita , gioco completato o meno, ecc.) a seconda delle quali vengono inviati vari eventi, alcuni dei quali illustrati precedentemente;

- *'chat'*: viene inviato quando viene premuto il button "*Send Message*" assieme al messaggio testuale presente nella casella di chat del giocatore. Alla ricezione di tale evento, il server rinvia ad entrambi gli utenti della sessione di gioco la coppia username mittente più messaggio ricevuto che verrà visualizzata nella chat dei giocatori.

Avendo descritto dettagliatamente il comportamento lato client e lato server, viene mostrato nell'immagine 1 un sequence diagram dell'interazione client-server durante una sessione di gioco.

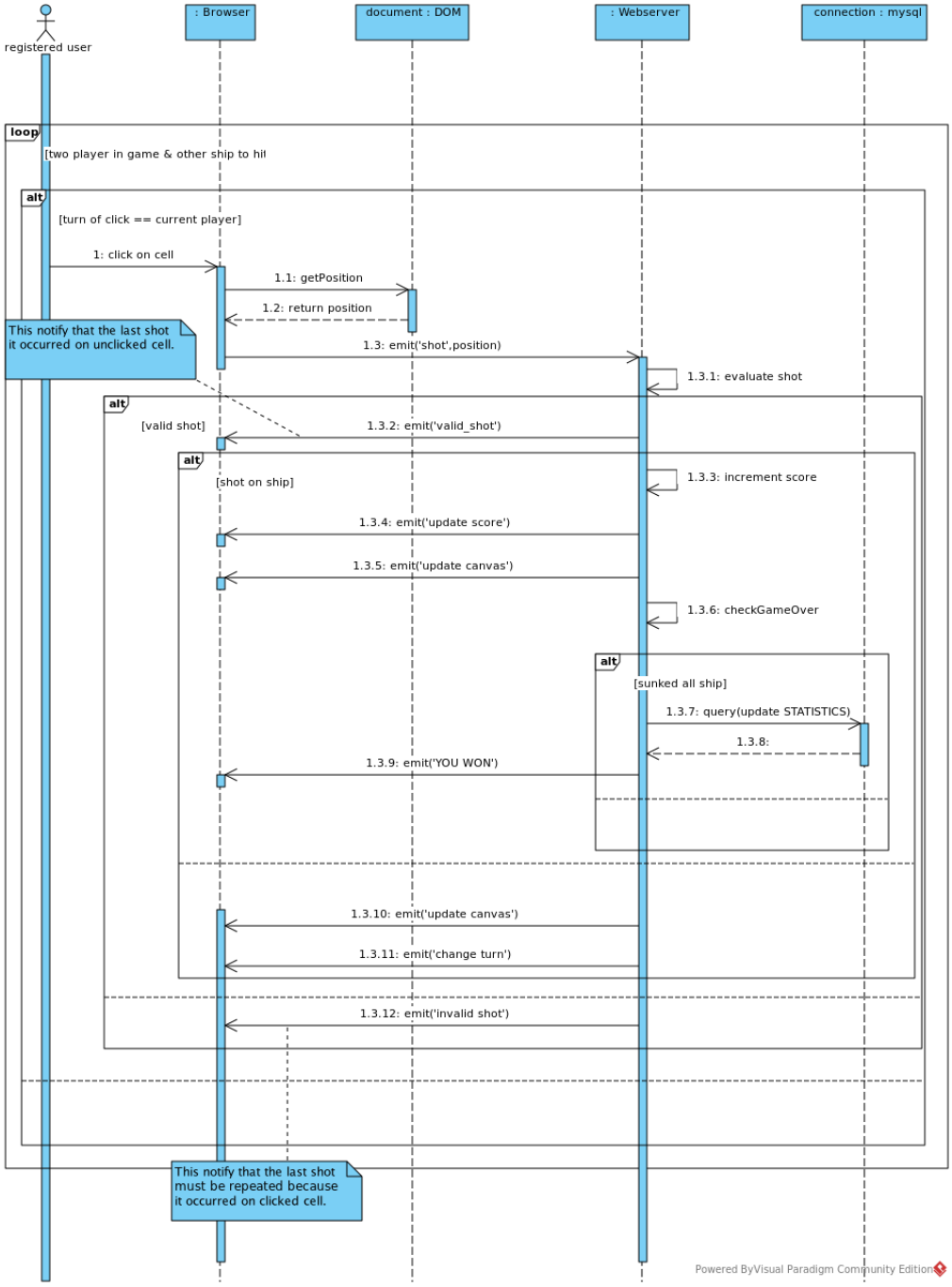

Figura 1: Sequence diagram interazione client-server

0.1 game.js

Il file **game.js** fa uso di **canvas**, un'estensione dell'HTML standard che permette il rendering dinamico di immagini bitmap gestibili attraverso un linguaggio di scripting.

Una delle funzionalità implementate più importanti è quella di realizzare la griglia dei giocatori attraverso le seguenti funzioni:

- *drawSquares()*: disegna le caselle della griglia;
- *drawShips()*: disegna le navi sulla griglia;
- *drawMarks()*: disegna sulla casella l'icona di una palla di fuoco in caso di nave colpita e l'icona di splash quando non viene colpita nessuna nave.

Quando il giocatore clicca una casella della griglia dell'avversario, vengono prese le coordinate del mouse al momento del click e viene richiamata la funzione *sendShot()*.

Algoritmo 4 Click event del giocatore su una casella avversaria

```
canvas[1].addEventListener('click', function(e) {
  if(turn) {
    var pos = getCanvasCoordinates(e, canvas[1]);
    var square = getSquare(pos.x, pos.y);
    sendShot(square);
  }
});
```