

# **Relatório Jogo Battleships em Sockets**

**Carina Emerim Leal, Felipe Emerim Leal**

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Sul  
(IFRS)  
92.412-240– Canoas – RS – Brazil

**Resumo.** *Este meta-artigo descreve a funcionalidade do jogo BattleShips, um jogo baseado em batalha naval porém com algumas particularidades que serão descritas ao longo deste documento. Serão apresentados também aspectos da linguagem utilizada para o desenvolvimento, conceitos de Sockets e protocolo TCP.*

## **1. Dinâmica Geral do Jogo BattleShips**

O jogo consiste em dois grids 10X10 onde em um deles o servidor aloca 9 barcos de maneira aleatória e em outro o Jogador aloca 9 barcos da maneira que desejar. Após o jogador e o servidor competem para ver quem encontra os 9 barcos escondidos pelo adversário primeiro. A cada jogada o servidor recebe as coordenadas enviadas pelo jogador, as processa e devolve a resposta, ou seja, se ele acertou um barco ou não e também a sua própria jogada no grid preenchido pelo jogador.

É possível jogar mais de um jogador ao mesmo tempo, porém não na mesma partida.

## **2. Protocolo TCP**

TCP é uma sigla que significa Transmission Control Protocol (Protocolo de Controle de Transmissões), que faz referência ao sistema de envio de pacotes mais comum da internet. Ao acessar um site, seu computador manda dados ao servidor pedindo que ele envie os conteúdos da página à máquina que está sendo utilizada, as informações enviadas de volta são interpretadas pelo seu navegador para mostrar aquilo que você deseja.

A principal característica do TCP é o fato de que ele não somente envia dados como também recebe informações de volta para se assegurar que os pacotes foram recebidos corretamente.

Na dinâmica do jogo BattleShips é importante existir essa confirmação para que seja retornado ao cliente se seu tiro acertou a água ou um barco, contabilizando assim seu score.

### 3. Sockets

Os sockets foram a forma de permitir que dois processos se comuniquem, esses processos podem ou não estar na mesma máquina.

Quando se programa utilizando Sockets, uma arquitetura muito comum para esses programas é utilizar o Cliente/Servidor, onde é necessário implementar um cliente e um servidor. Ambos utilizam a mesma API de Sockets.

No geral existem dois tipos de sockets: TCP e UDP. Os dois tipos são controlados pela API de sockets de maneira a abstrair detalhes da rede para o desenvolvedor. Ela está imediatamente acima da camada de transporte e abaixo da camada de aplicação.

Os sockets do tipo TCP são orientados a conexão e tem um canal exclusivo de comunicação entre cliente e servidor. Eles garantem a ordem dos pacotes, são considerados confiáveis e sem perda.

Já os sockets do tipo UDP desconsideram ordem de pacotes, recuperação de falhas e garantia de ordem. No entanto, por ser extremamente menos burocrático e simples, ele é mais rápido que o TCP para alguns tipos de aplicações.

Temos um exemplo utilizando socket.io no jogo Battleships, onde o servidor recebe a posição do tiro do jogador através de JSON e devolve o resultado, ou seja, se o tiro foi na água ou em um barco (Imagem 9).

### 4. JavaScript

O jogo BattleShips foi desenvolvido em JavaScript, que é uma linguagem que permite “dar vida” a uma página web, ou seja, alterar a página reagindo a eventos emitidos pelo usuário. O servidor foi desenvolvido usando node, express e socket.io.

Node.js é uma plataforma construída sobre o motor JavaScript do Google Chrome para facilmente construir aplicações de rede rápidas e escaláveis. Node.js usa um modelo de I/O direcionada a evento não bloqueante que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos. O express e o socket.io são frameworks do Node, o express possui um conjunto de recursos para desenvolver aplicações Web, como um sistema de Model, View And Controller(MVC). Socket.io é uma biblioteca Javascript feita para construir aplicações real-time, possibilitando uma comunicação bi-direcional entre cliente e servidor.

## **5. Funcionamento Jogo BattleShips**

A primeira coisa a ser feita é instalar o node js se não possuir. Após isto é necessário instalar as dependências através do NPM(Node Package Manager) que vem junto com o node. O próximo passo é inicializar o servidor Socket, rodando o Index.js (Imagem 1). Após, o cliente precisa acessar este servidor informando o IP do servidor e a porta na qual o servidor está ouvindo. É necessário ter conhecimento prévio sobre a porta do Servidor que está ouvindo (Listen). (Imagem 2).

Ao estabelecer a conexão(Imagem 3) o servidor retorna dois grids onde em um deles o Jogador deverá setar os barcos para que o servidor jogue, e no outro acontecerá o contrário. (Imagem 4)

Para jogar o cliente deverá clicar em um dos quadrados que constam no grid preenchido pelo servidor (Imagem 5).

Quem localizar os nove barcos primeiro ganha o jogo (imagem 6).

## **6. WireShark**

WireShark, conhecido como tubarão dos fios, serve para monitorar os pacotes de informações que trafegam através de sua rede, um analisador de protocolos para redes de computadores e, no momento, é considerado um dos mais utilizados para Linux no momento, desenvolvido pela Ethereal.

É uma ferramenta totalmente livre, ou seja, você pode baixá-la e não precisa se preocupar com limitações ou prazo de validade, apenas instalar e sair usando.

O administrador da rede, ou o responsável pela rede pode ter o controle de tudo o que entra e sai da rede, em diferentes protocolos.

Uma boa opção para quem tem uma grande rede para administrar, porque cuidar de uma rede pequena não é tão complicado, mas quando falamos sobre uma grande empresa, a visão já é diferente.

Utilizamos o Wireshark para verificar o comportamento dos pacotes enquanto o jogo BattleShips é iniciado. (Imagem 7 e 8).

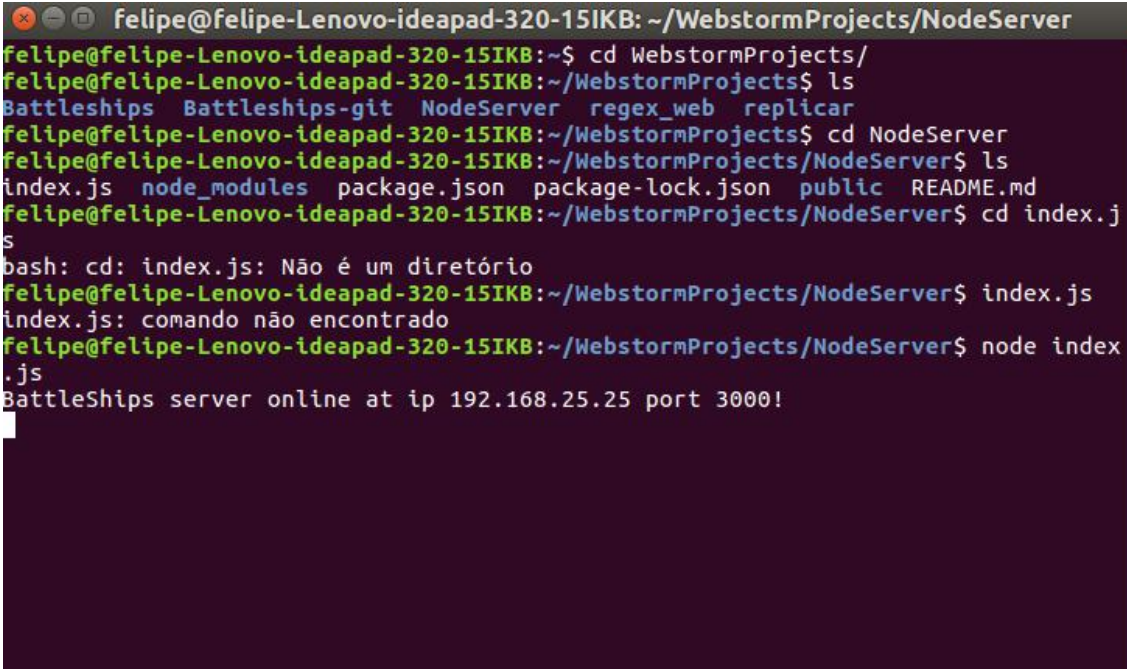
## 7. Conclusão

Ao longo do desenvolvimento deste trabalho conseguimos entender melhor alguns conceitos vistos em aula, principalmente sobre utilização de servidor e sua comunicação com o cliente.

Pudemos observar como funciona a troca de mensagens entre cliente-servidor e contextualizar alguns conceitos como protocolo TCP e UDP, socket, JSON e também aspectos da linguagem JavaScript em geral.

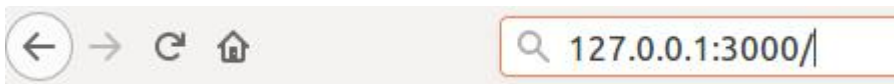
Iniciamos o desenvolvimento em Java, porém decidimos criar uma interface mais amigável ao cliente utilizando html, css e JavaScript no front-end, enquanto o back-end foi desenvolvido em JavaScript utilizando os frameworks Node, express e socket.io.

## 8. Imagens

A terminal window with a dark background and light green text. The window title is 'felipe@felipe-Lenovo-ideapad-320-15IKB: ~/WebstormProjects/NodeServer'. The user navigates through directories and runs a Node.js script. The final output is 'BattleShips server online at ip 192.168.25.25 port 3000!'.

```
felipe@felipe-Lenovo-ideapad-320-15IKB: ~/WebstormProjects/NodeServer
felipe@felipe-Lenovo-ideapad-320-15IKB:~$ cd WebstormProjects/
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects$ ls
Battleships  Battleships-git  NodeServer  regex_web  replicar
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects$ cd NodeServer
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects/NodeServer$ ls
index.js  node_modules  package.json  package-lock.json  public  README.md
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects/NodeServer$ cd index.j
S
bash: cd: index.js: Não é um diretório
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects/NodeServer$ index.js
index.js: comando não encontrado
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects/NodeServer$ node index
.js
BattleShips server online at ip 192.168.25.25 port 3000!
```

Images 1. Autoria Própria



Images 2. Autoria Própria

```
felipe@felipe-Lenovo-ideapad-320-15IKB: ~/WebstormProjects/NodeServer
felipe@felipe-Lenovo-ideapad-320-15IKB:~$ cd WebstormProjects/
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects$ ls
Battleships Battleships-git NodeServer regex_web replicar
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects$ cd NodeServer
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects/NodeServer$ ls
index.js node_modules package.json package-lock.json public README.md
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects/NodeServer$ cd index.j
s
bash: cd: index.js: Não é um diretório
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects/NodeServer$ index.js
index.js: comando não encontrado
felipe@felipe-Lenovo-ideapad-320-15IKB:~/WebstormProjects/NodeServer$ node index
.js
BattleShips server online at ip 192.168.25.25 port 3000!
::ffff:192.168.25.16 connected
```

Images 3. Autoria Própria



Imagem 4. Autoria Própria



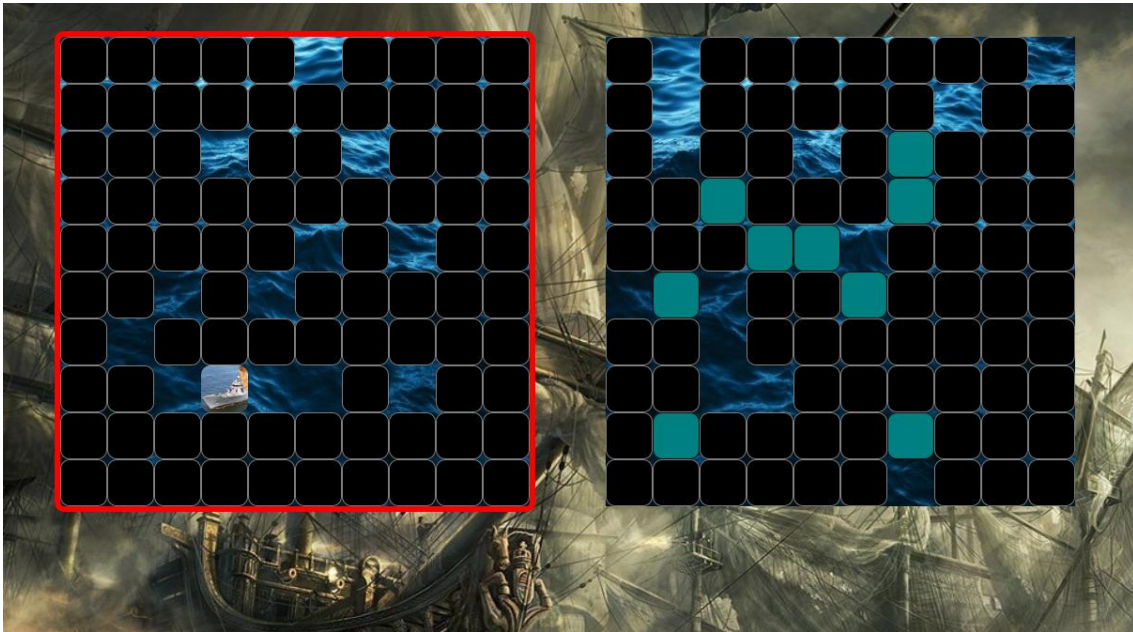


Imagem 5. Autoria Própria

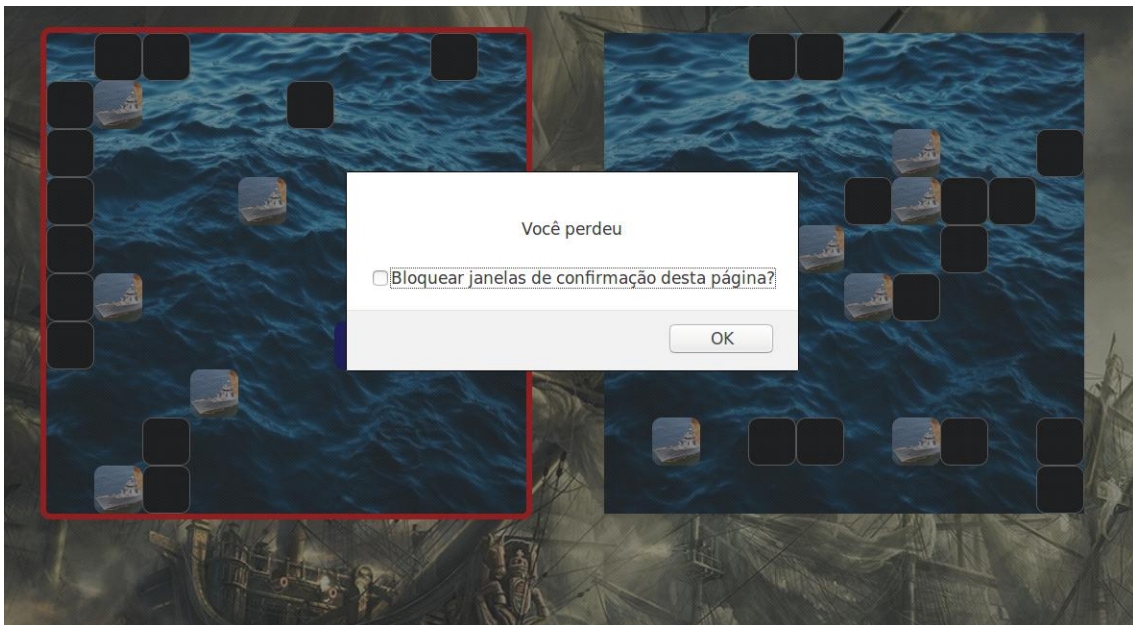


Imagem 6. Autoria Própria

No.	Time	Source	Destination	Protocol	Length	Info
6	5.524681593	127.0.0.1	127.0.0.1	TCP	54	80 → 47214 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
9	21.065488168	127.0.0.1	127.0.0.1	TCP	74	57396 → 3000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=2314025334 TSecr=0 WS=128
10	21.065498038	127.0.0.1	127.0.0.1	TCP	74	3000 → 57396 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=2314025334 TSecr=2314025334
11	21.065505229	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=2314025334 TSecr=2314025334
13	21.065594454	127.0.0.1	127.0.0.1	TCP	66	3000 → 57396 [ACK] Seq=1 Ack=529 Win=44800 Len=0 TSval=2314025334 TSecr=2314025334
15	21.074338939	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=529 Ack=243 Win=44800 Len=0 TSval=2314025343 TSecr=2314025343
18	21.130115958	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=1031 Ack=486 Win=45952 Len=0 TSval=2314025399 TSecr=2314025399
22	21.154335346	127.0.0.1	127.0.0.1	TCP	74	57396 → 3000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=2314025423 TSecr=0 WS=128
23	21.154382661	127.0.0.1	127.0.0.1	TCP	74	3000 → 57396 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=2314025423 TSecr=2314025423
24	21.154423109	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=2314025423 TSecr=2314025423
26	21.201711129	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=1949 Ack=819 Win=46076 Len=0 TSval=2314025423 TSecr=2314025423
28	21.223673917	127.0.0.1	127.0.0.1	TCP	66	3000 → 57396 [ACK] Seq=1 Ack=430 Win=44800 Len=0 TSval=2314025492 TSecr=2314025492
30	21.229743714	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=430 Ack=333 Win=44800 Len=0 TSval=2314025498 TSecr=2314025498
33	21.260388254	127.0.0.1	127.0.0.1	TCP	74	57400 → 3000 [SYN] Seq=0 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=2314025529 TSecr=0 WS=128
34	21.260395205	127.0.0.1	127.0.0.1	TCP	74	3000 → 57400 [SYN, ACK] Seq=0 Ack=1 Win=43690 Len=0 MSS=65495 SACK_PERM=1 TSval=2314025529 TSecr=2314025529
35	21.260403615	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=1 Ack=1 Win=43776 Len=0 TSval=2314025529 TSecr=2314025529
37	21.260505512	127.0.0.1	127.0.0.1	TCP	66	3000 → 57400 [ACK] Seq=1 Ack=720 Win=45184 Len=0 TSval=2314025529 TSecr=2314025529
41	21.265518501	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=720 Ack=176 Win=44800 Len=0 TSval=2314025534 TSecr=2314025534
47	21.301769063	127.0.0.1	127.0.0.1	TCP	66	3000 → 57396 [ACK] Seq=819 Ack=2428 Win=49152 Len=0 TSval=2314025529 TSecr=2314025529
48	21.317798936	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=2129 Ack=1004 Win=48000 Len=0 TSval=2314025540 TSecr=2314025540
49	21.317800499	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=734 Ack=184 Win=44800 Len=0 TSval=2314025539 TSecr=2314025539
52	21.323991899	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=2711 Ack=1249 Win=49152 Len=0 TSval=2314025593 TSecr=2314025593
56	21.372171748	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=2428 Ack=1049 Win=48000 Len=0 TSval=2314025641 TSecr=2314025641
58	21.381760586	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=3292 Ack=1494 Win=50176 Len=0 TSval=2314025609 TSecr=2314025609
59	21.413774648	127.0.0.1	127.0.0.1	TCP	66	3000 → 57400 [ACK] Seq=184 Ack=743 Win=45184 Len=0 TSval=2314025642 TSecr=2314025642
62	21.723169593	127.0.0.1	127.0.0.1	TCP	66	3000 → 57396 [ACK] Seq=1049 Ack=2876 Win=50176 Len=0 TSval=2314025992 TSecr=2314025992
64	21.757150381	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=3568 Ack=1711 Win=51200 Len=0 TSval=2314026026 TSecr=2314026026
66	21.762476019	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=2876 Ack=1288 Win=49152 Len=0 TSval=2314026031 TSecr=2314026031
68	21.803854439	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [ACK] Seq=4091 Ack=1953 Win=52352 Len=0 TSval=2314026072 TSecr=2314026072
70	26.764004872	127.0.0.1	127.0.0.1	TCP	66	3000 → 57396 [FIN, ACK] Seq=1288 Ack=2876 Win=50176 Len=0 TSval=2314031031 TSecr=2314026031
71	26.764108382	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [FIN, ACK] Seq=2876 Ack=1289 Win=49152 Len=0 TSval=2314031032 TSecr=2314031031
72	26.764119046	127.0.0.1	127.0.0.1	TCP	66	3000 → 57396 [ACK] Seq=1289 Ack=2877 Win=50176 Len=0 TSval=2314031032 TSecr=2314031032
73	26.805381170	127.0.0.1	127.0.0.1	TCP	66	3000 → 57396 [FIN, ACK] Seq=1953 Ack=4091 Win=52864 Len=0 TSval=2314031073 TSecr=2314026072
74	26.805670785	127.0.0.1	127.0.0.1	TCP	66	57396 → 3000 [FIN, ACK] Seq=4091 Ack=1954 Win=52352 Len=0 TSval=2314031073 TSecr=2314031073
75	26.805702825	127.0.0.1	127.0.0.1	TCP	66	3000 → 57396 [ACK] Seq=1954 Ack=4092 Win=52864 Len=0 TSval=2314031073 TSecr=2314031073
77	46.252416746	127.0.0.1	127.0.0.1	TCP	66	3000 → 57400 [ACK] Seq=184 Ack=752 Win=45184 Len=0 TSval=2314050515 TSecr=2314050515

Imagem 7. Testes WireShark Autoria Própria

No.	Time	Source	Destination	Protocol	Length	Info
74	26.805670785	127.0.0.1	127.0.0.1	TCP	66	57398 → 3000 [FIN, ACK] Seq=4091 Ack=1954 Win=52352 Len=0 TSval=2314031073 TSecr=2314031073
75	26.805702825	127.0.0.1	127.0.0.1	TCP	66	3000 → 57398 [ACK] Seq=1954 Ack=4092 Win=52864 Len=0 TSval=2314031073 TSecr=2314031073
77	46.252416746	127.0.0.1	127.0.0.1	TCP	66	3000 → 57400 [ACK] Seq=184 Ack=752 Win=45184 Len=0 TSval=2314050515 TSecr=2314050515
79	46.252873796	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=752 Ack=187 Win=44800 Len=0 TSval=2314050516 TSecr=2314050516
82	71.256116278	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=761 Ack=190 Win=44800 Len=0 TSval=2314075512 TSecr=2314075512
42	21.266126889	127.0.0.1	127.0.0.1	WebSoc...	80	WebSocket Text [FIN] [MASKED]
45	21.270709874	127.0.0.1	127.0.0.1	WebSoc...	74	WebSocket Text [FIN]
57	21.373563650	127.0.0.1	127.0.0.1	WebSoc...	75	WebSocket Text [FIN] [MASKED]
76	46.252406995	127.0.0.1	127.0.0.1	WebSoc...	75	WebSocket Text [FIN] [MASKED]
78	46.252870666	127.0.0.1	127.0.0.1	WebSoc...	69	WebSocket Text [FIN]
80	71.254125136	127.0.0.1	127.0.0.1	WebSoc...	75	WebSocket Text [FIN] [MASKED]
81	71.256082784	127.0.0.1	127.0.0.1	WebSoc...	69	WebSocket Text [FIN]
83	96.258936952	127.0.0.1	127.0.0.1	WebSoc...	75	WebSocket Text [FIN] [MASKED]
84	96.260308673	127.0.0.1	127.0.0.1	WebSoc...	69	WebSocket Text [FIN]
85	96.260343553	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=770 Ack=193 Win=44800 Len=0 TSval=2314100515 TSecr=2314100515
86	121.262875602	127.0.0.1	127.0.0.1	WebSoc...	75	WebSocket Text [FIN] [MASKED]
87	121.263474329	127.0.0.1	127.0.0.1	WebSoc...	69	WebSocket Text [FIN]
88	121.263488148	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=779 Ack=196 Win=44800 Len=0 TSval=2314125526 TSecr=2314125526
89	133.084394682	127.0.0.1	127.0.0.1	DNS	77	Standard query 0xcd05 A www.jetbrains.com
90	133.275209953	127.0.0.1	127.0.0.1	DNS	215	Standard query response 0xcd05 A www.jetbrains.com CNAME www-weighted.jetbrains.com CNAME w3jbcom-pr...
91	134.533893096	127.0.0.1	127.0.0.1	DNS	81	Standard query 0x8082 A account.jetbrains.com
92	134.608818940	127.0.0.1	127.0.0.1	DNS	181	Standard query response 0x8082 A account.jetbrains.com CNAME jetprofile-prod-alb-648241618.eu-west-1...
93	146.266194118	127.0.0.1	127.0.0.1	WebSoc...	75	WebSocket Text [FIN] [MASKED]
94	146.267923561	127.0.0.1	127.0.0.1	WebSoc...	69	WebSocket Text [FIN]
95	146.267969558	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=788 Ack=199 Win=44800 Len=0 TSval=2314150534 TSecr=2314150534
96	149.357498565	127.0.0.1	127.0.0.1	DNS	84	Standard query 0x6686 A notifications.google.com
97	149.394821869	127.0.0.1	127.0.0.1	DNS	121	Standard query response 0x6686 A notifications.google.com CNAME plus.l.google.com A 172.217.29.206
98	171.270211276	127.0.0.1	127.0.0.1	WebSoc...	75	WebSocket Text [FIN] [MASKED]
99	171.270646541	127.0.0.1	127.0.0.1	WebSoc...	69	WebSocket Text [FIN]
100	171.270657929	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=797 Ack=202 Win=44800 Len=0 TSval=2314175537 TSecr=2314175537
101	196.272046406	127.0.0.1	127.0.0.1	WebSoc...	75	WebSocket Text [FIN] [MASKED]
102	196.272520128	127.0.0.1	127.0.0.1	WebSoc...	69	WebSocket Text [FIN]
103	196.272529290	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=806 Ack=205 Win=44800 Len=0 TSval=2314200538 TSecr=2314200538
104	221.274039683	127.0.0.1	127.0.0.1	WebSoc...	75	WebSocket Text [FIN] [MASKED]
105	221.274709918	127.0.0.1	127.0.0.1	WebSoc...	69	WebSocket Text [FIN]
106	221.274724438	127.0.0.1	127.0.0.1	TCP	66	57400 → 3000 [ACK] Seq=815 Ack=208 Win=44800 Len=0 TSval=2314225537 TSecr=2314225537

▶ Frame 94: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface 0  
 ▶ Ethernet II, Src: 00:00:00:00:00:00 (00:00:00:00:00:00), Dst: 00:00:00:00:00:00 (00:00:00:00:00:00)  
 ▶ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1  
 ▶ Transmission Control Protocol, Src Port: 57396, Dst Port: 3000, Seq: 0, Len: 0

Imagem 8. Testes WireShark Autoria Própria



```

socket.on('user-shot', function(data){ //evento que valida o tiro do usuario,
//recebe coordenadas e id do elemento através de JSON
if(typeof(socket['campo']) === "undefined"){
    socket.emit('refresh-page');
}
else {
    switch (socket['cpuCampo'][data.s_line][data.s_column]) {
        case 0: //tiro na água
            socket['cpuCampo'][data.s_line][data.s_column] = 2; //seta célula para tiro na água
            socket.emit('player-water-shot', {id: data.id}); //emite tiro na água para o cliente
            break;

        case 1: //tiro em barco
            socket['cpuCampo'][data.s_line][data.s_column] = 3; //seta célula para tiro em barco
            socket['hits']++; //aumenta o contador de acertos do cliente
            socket.emit('player-boat-shot', {id: data.id}); //emite tiro em barco para o cliente
            break;

        default: //tiro inválido
            socket.emit('wasted-shot'); //emite evento de tiro inválido para o cliente
    }
}
});

```

**Imagem 9. Autoria Própria**

## 9. References

SAUDE, Pedro. O protocolo TCP. Disponível em: <<https://br.ccm.net/contents/284-o-protocolo-tcp>>. Acesso em: 25 maio 2018.

PINTO, Pedro. Redes: Quais as diferenças entre o Protocolo TCP e UDP. Disponível em: <<https://pplware.sapo.pt/tutoriais/redes-quais-diferencas-protocolo-tcp-udp/>>. Acesso em: 25 maio 2018.

SILVEIRA, Cristiano Bertulucci. Entenda como funciona o Protocolo TCP-IP. Disponível em: < <https://www.citisystems.com.br/protocolo-tcp-ip/>>. Acesso em: 25 maio 2018.

ALMEIDA, Thalisson Christiano de. **Desenvolvimento de jogos em rede:** Camada OSI e protocolo TCP. Disponível em: < <http://www.fabricadejogos.net/posts/desenvolvimento-de-jogos-em-rede-camada-osi-e-protocolo-tcp/>>. Acesso em: 25 maio 2018.



PINTO, Pedro. Redes - Sabe o que são sockets de comunicação? Disponível em: < <https://pplware.sapo.pt/tutoriais/networking/redes-sabe-o-que-sao-sockets-de-comunicacao-parte-i/>>. Acesso em: 25 maio 2018.

NODEJS - Disponível em: < <https://nodejs.org/en/>>. Acesso em: 01 junho 2018.

CAMPOS, Daniel. Criando uma aplicação de Chat simples com NodeJS e Socket.io. Disponível em: <<https://tableless.com.br/criando-uma-aplicacao-de-chat-simples-com-nodejs-e-socket-io/>>. Acesso em: 01 junho 2018.

Primeiros passos com Express em node.js. Disponível em: < <http://nodebr.com/primeiros-passos-com-express-em-node-js/>>. Acesso em: 01 junho 2018.

Blog Wireshark. O que é Wireshark? Disponível em: < <https://rodrigobastos.wordpress.com/2011/06/12/o-que-e-o-wireshark/>>. Acesso em: 06 junho 2018.