Trabalho Prático 2 – Battleship Board Game

Josué Ferreira (nº 2066513) and Pedro Rodrigues (nº 2091412)

Univsersidade da Madeira (UMa), Funchal, Portugal
yoshuaferreira@gmail.com
pedroagrodrigues@gmail.com

Abstrato. Este relatório aborda o desenvolvimento do jogo de tabuleiro Batalha Naval, como objetivo da unidade curricular de Aplicações Centradas em Redes (ACR), uma aplicação jogável entre dois utilizadores através da utilização de *sockets* de rede e sessões. O desenvolvimento desta aplicação realizou-se recorrendo à programação em Javascript para o *back-end* e utiliza, para além do HTML, *Embedded JavaScript* e Vue.js.

Keywords: Battleship, Javascript, Vue, Express, Socket.io.

1 Introdução

No âmbito da unidade curricular académica de Aplicações Centradas em Redes, foi proposto aos alunos o desenvolvimento de uma aplicação de rede. O tema desta aplicação, baseia-se num jogo de tabuleiro, comercializado e publicado pela *Milton Bradley Company* em 1931, denominado de Batalha Naval [1].

O objetivo deste projeto é introduzir o conceito de *full-stack-development* aos alunos.

Durante o desenvolvimento desta aplicação tiveram de ser respeitados vários requisitos

1.1 Requisitos de implementação

Os alunos tinham que cumprir os requisitos tecnológicos abaixo descritos:

- Utilizar uma biblioteca (cliente e servidor) *socket.io*
- Recorrendo ao *template engine* EmbedJS (.ejs como extensão de formato)
- Utilizar a framework Vue.js para o lado do cliente
- Utilizar a framework *express* para o lado do servidor

Para os requisitos funcionais:

- Permitir registo dos jogadores
- Permitir o registo dos jogos
- Permitir com que guarde e carregue um jogo que não foi finalizado e que apenas são permitidos os jogadores que participaram essa sessão.
- A comunicação durante o jogo e indicar visualmente o estado do jogo

2 Problema

Durante o desenvolvimento do projeto proposto, decidimos tomar conhecimento às regras que devem ser aplicadas no jogo e criar um conjunto de ideias que imita e compatibiliza e o funcionamento do mesmo.

Quais são as regras principais? Os dois jogadores posicionam os seus barcos num tabuleiro quadrangular de 9 por 9. Depois de todas as peças estarem posicionadas, os jogadores tentam, à vez, adivinhar a posição dos barcos do adversário de modo a atingir os mesmos. Os jogadores comunicam restritamente como: "atingiu", "falhou" e "afundou o barco". As tentativas são representadas por marcadores vermelhos no caso de acertar na posição do navio adversário e por marcadores brancos quando falha. Opcionalmente, existe um modo avançado onde os jogadores disparam cinco vezes por cada turno, cada vez que o jogador perde um dos seus barcos subtrai por um valor o número de disparos por cada turno[8].

2.1 Implementação de ideias

Após a observação do funcionamento do jogo, tomamos a decisão de tentar imitar a implementação do jogo usando a linguagem de programação Javascript. Recorremos ao Vue.js para criar dois tabuleiros que identificam o jogador e o adversário. De modo a representar as zonas onde se atingiu ou falhou utilizou-se CSS de maneira a mudar a cor da quadrícula escolhida para vermelho ou azul consoante acerta ou falha.

Durante o desenvolvimento do *front-end* implementou-se um botão recorrendo a CSS3 com a funcionalidade de implementar a funcionalidade *Dark* Theme, a tendência visual que seguem de vários programas atualmente.

2.2 Influências de trabalho

Para além de recorrer ao método exemplo, como foi fornecido no enunciado deste projeto para aplicar o sistema base de chat online, recorremos a pesquisa de vários repositórios que contenha alguns dos requisitos pedidos, para que possamos aplicar *reverse engineering* (engenharia reversa) para a entender o funcionamento deste e construir o nosso próprio jogo.

Por este motivo, creditou-se os autores desses repositórios e de outros conteúdos que se interligam a este projeto nas Referências[5,6, 7].

2.3 Instalação de pacotes

Recorremos à ferramenta *Node Package Manager* para a instalação de pacotes necessários de maneira a cumprir os requisitos tecnológicos que foram pedidos no enunciado do projeto.

Pacotes Instalados:

- Socket.io (uma biblioteca que permite comunicação web),
- *template engine* Embedded JS,
- framework *Vue.js* (para o lado do cliente),
- framework *express* (para o lado do servidor).
- cookie-parser,
- express-session
- express-session.io-session.
- Cookie-parser que grava as informações temporariamente.
- Express-session que possibilita criação de múltiplas sessões.
- Express-session.io-session partilha uma cookie do express com a socket.io.

3 Implementação de Requisitos

3.1 Base de dados

Primeiramente, foi necessário implementar uma base de dados para registar os jogadores e os jogos. A tabela *user*, recorremos aos campos essenciais são: email, nome e password (de forma encriptada), para a tabela *game* contém as informações dos jogadores (*player1* e *player2*), o estado do jogo (*gameState*), o *shipPlayer1* e o *shipPlayer2*, estes últimos recorre a *longtext* para obter vários caracteres para informações sobre o jogo realizado. Decidimos recorrer à base de dados *MySQL* (*XAMPP/standalone*), uma vez que estamos acostumados com esta linguagem de *query SQL*, desde a unidade curricular Sistemas de Gestores e Base de Dados.

3.2 Login e Registo

Na parte de *Front-End* para o *Login* e o *Register*, recorremos às bases do trabalho prático P1 – *"Knight of Gamers"*, caso houvesse campos por preencher ativa um aviso para preencher os campos que faltavam ou falha de autenticação.

Criou-se uma atribuição de variáveis de sessão para que múltiplas sessões pudessem ser criadas, tornando o servidor *multi-user*.

3.3 Banner

Antes de passarmos ao funcionamento da página principal, foi implementado uma barra de navegação simples que contém o Login/Register, identifica os utilizadores que acedem ao site, se nunca registou no site identifica-o como "Guest", juntamente com o valor da *socket* que identifica o jogador. Outra implementação que foi inserida é o Dark Theme. No momento que interage, não só ambiente muda de cor mas, como também há uma mudança de imagens entre Sol e Lua no botão que o utilizador interage.

3.4 Página Principal

A página principal contém ligações de múltiplos ficheiros *EmbedJS* (banner.ejs, index.ejs e game.ejs). Contém duas tabelas que representa o jogador e o adversário, na tabela do jogador posiciona os barcos, enquanto na tabela do adversário o jogador escolhe as posições para atingir os barcos do adversário.

A escolha de posições é efetuada através da interação do rato, no momento que aproximamos à nossa tabela, juntamente com o barco escolhido, cria um pré-preenchimento na tela, marcando quantos lugares ocupará numa determinada posição. Caso tenta posicionar o barco numa zona ocupada por um ou vários (que estejam lado a lado), o barco não será posicionado. Portanto, o utilizador terá que escolher outra posição. Clicando no lado direito do rato, faz com que faça rotação para a vertical o barco.

Como o sistema é gerido por *sockets* através da *socket.io*, o jogo só pode continuar se tiver um oponente e se já posicionou os barcos. Quando estiver tudo a postos para o segundo jogador, através destas *sockets* o servidor fornece a permissão para começar o jogo, indicando mensagens entre eles.

Uma sessão só pode ser feita apenas com dois jogadores através da interação com o botão "*Join Room*", quando começa o jogo indica as mensagens "*Not Ready*" e "*Room name*" com o valor da sessão atual. Se um dos jogadores sair por algum propósito ou se teve uma falha de ligação a sessão é interrompida e a ronda deixa de existir.

Na política de turnos foi tomada a decisão que o primeiro jogador a estar pronto deve disparar primeiro, para evitar qualquer vulnerabilidade ou causar *miss shot* por parte de um dos jogadores antes de começar a batalha naval. Para os jogadores tomarem conhecimento dos seus turnos como "*Your turn*" e "*Enemy's turn*", todas estas mudanças eram efetuadas atualizadas através da extensão socket.io. Quando o utilizador atinge uma posição do adversário ou sofre um ataque por parte do adversário, indicará a mudança de cor na tela. Se atingir a tela fica vermelha, se falhar a tela fica azul.

Apenas existem duas maneiras para finalizar um jogo: ganhar ou perder, funciona através de contagem de blocos que foram atingidos com o número total de dezassete blocos, ou seja, no jogo não determina quantos barcos foram afundados.

3.5 Requisitos em falta

Enquanto era desenvolvido a aplicação, houve limitações para concluir certos requisitos necessários por parte do projeto. Apesar de existir uma funcionalidade única onde cada jogador acede a uma sessão única com outro jogador, não há efeito que os jogadores consigam guardar as informações dos jogos, caso os jogadores querem interromper o jogo. Ou seja, não guardar as informações do jogo para puder continuar de futuro, devido uma falha interna que não contém o adversário na sessão.

4 Conclusão

Embora consideremos que a utilização do *Vue* em conjunto com *EJS* seja desnecessária, concluímos que ao usar este tipo de *frameworks* trás imensas vantagens. Depois da elaboração deste trabalho conseguimos compreender melhor o que significa *full-stack* e concordamos que muitos dos desafios e problemas ultrapassados neste trabalho contribuíram para o nosso desenvolvimento pessoal.

Consideramos importante a utilização destas *frameworks* e outras semelhantes, o desenvolvimento de aplicações torna-se mais simples e atualizado e facilita muito no caso que tenhamos de desenvolver mais aplicações semelhantes.

Referências

1. ‘Batalha naval (jogo)’. Wikipédia, a enciclopédia livre, 27 Oct. 2019. Wikipedia, https://pt.wikipedia.org/w/index.php?title=Batalha_naval_(jogo)&oldid=56575820.
2. ‘Stack Overflow - Where Developers Learn, Share, & Build Careers’. Stack Overflow, https://stackoverflow.com/. Accessed 13 Jan. 2020.
3. YouTube. https://www.youtube.com/. Accessed 13 Jan. 2020.
4. GitHub - http://github.com
5. Arend. Zasei/Battleship. 2016. 2019. GitHub, https://github.com/zasei/Battleship.
6. Nilsson, Christian. Inf123/NodeBattleship. 2015. 2019. GitHub, https://github.com/inf123/NodeBattleship.
7. Romano. Romanornr/Battleship-Nodejs. 2016. 2019. GitHub, https://github.com/romanornr/Battleship-Nodejs.
8. ‘Learn the Basics of Battleship with This Easy Guide’. The Spruce Crafts, https://www.thesprucecrafts.com/the-basic-rules-of-battleship-411069. Accessed 13 Jan. 2020.