Battle Sea JS

Raed Abdennadher et Steven Liatti

Université d'été - Prof Jean-Luc Falcone

Hepia ITI 2ème année

11 septembre 2017

1 Introduction

1.1 Description

Le but de notre projet est d'implémenter un jeu de bataille navale ("touché-coulé") en Javascript en réseau. Le jeu de bataille navale est un jeu tour par tour ou chaque joueur dispose des bateaux sur une grille. Le but du jeu est de couler tous les bateaux de son adversaire. Le projet est disponible sur https://github.com/steenput/battleseajs

1.2 Cahier des charges

Nous avons créé le mode de jeu standard, avec une interface graphique (le positionnement des bateaux manuel ou aléatoire, cliquer pour "toucher") côté client. Les bateaux sont représentés par des images. Coté serveur, nous faisons le lien entre les joueurs (transmission des ordres de tir, répercussion sur la grille de l'adversaire, système "d'événements" pour afficher des informations). Un mode joueur contre ordinateur a également été implémenté. L'application est multi instance, plusieurs parties peuvent se jouer en simultané.

1.3 Répartition du travail

Raed s'est occupé de l'interface graphique. Steven s'est occupé du serveur : l'échange des événements et le mode de jeu contre l'ordinateur.

1.4 Technologies choisies

En plus du Javascript "pur" (ES6), nous avons utilisé les technologies suivantes :

- Côté client :
  - Data-Driven Documents pour la majeur partie de l'interface "cliquable"
  - un peu de jQuery
  - Bootstrap V4 pour le design en CSS
- Côté serveur :
  - Node.js, avec
  - socket.io pour l'échange dynamique des informations et
  - Express pour gérer les requêtes classiques (peu utilisé)

2 Réalisation de Raed

2.1 Buts

2.2 Travail réalisé

2.3 (Questions -) Réponses

3 Réalisation de Steven

3.1 Buts

Mon but était de mettre en place un serveur avec Node.js pour gérer la communication entre les joueurs et implémenter un mode de jeu "joueur contre ordinateur" (dans un second temps).

3.2 Travail réalisé

J'ai commencé par étudier Node.js. J'avais besoin d'un mécanisme d'échange d'informations sans devoir forcément recharger la page entière. Je pensais aux sockets que j'avais étudié en C en cours de système d'exploitation. J'ai également pensé à AJAX. En étudiant son fonctionnement et en lisant plusieurs documentations/tutoriels, j'ai découvert la librairie socket.io qui correspondait exactement à ce dont j'avais besoin. Après quelques essais et test, nous l'avons pleinement adoptée pour communiquer entre le client et le serveur. J'ai posé la structure des entités du jeu (dans model.js) et je l'ai utilisée dans server.js pour les étapes successives du jeu : création/adjonction d'une partie, placement des bateaux sur la grille, échange de tirs, destruction de bateau, fin du jeu. Une fois que le mode de jeu "joueur contre joueur" était fonctionnel, je me suis attelé au mode "joueur contre ordinateur". J'ai été agréablement surpris par le fait que socket.io fournit également son API côté client, j'ai donc pu réutiliser le format des événements déjà implémentés. Il m'a fallu gérer le positionnement aléatoire des bateaux de l'ordinateur et son style de jeu aléatoire également.

3.3 (Questions -) Réponses

Qu'est ce qui vous a positivement surpris ? L'efficacité et la simplicité de la librairie socket.io : une fois les bases comprises, l'usage est très facile et chouettes sont les possibilités.

Qu'est ce qui vous a pris plus de temps ? La définition du modèle et se mettre d'accord sur le format d'échange de données entre client-serveur.

Quel est le pire bug que vous avez eu ? Je n'ai pas personnellement souvenir d'un bug complètement monstrueux. De manière générale, au début du développement, je dirai le langage en lui-même : il n'était pas toujours clair de savoir pourquoi une instruction ne donnait pas le comportement attendu (syntaxe du langage).

Au final si vous deviez recommencer que changeriez-vous ? Je prendrai plus de temps pour peaufiner l'API publique et dès le départ fixer son format et ses usages (côté serveur pour ma part).

4 Conclusion

4.1 État actuel du projet

4.2 Propositions d'améliorations

— Côté client :

  - Enrichir l'interface graphique : effets d'explosion, animations, etc.
  - Ajout des effets sonores.
  - Modes de jeu alternatifs (que le classique).
- Côté serveur :
  - Mode spectateur : identifiant unique par partie qui permet de la visionner.
  - Enregistrement en base de données (MongoDB) pour rejouer les parties.