Battle Sea JS

Raed Abdennadher et Steven Liatti

Université d'été - Prof Jean-Luc Falcone

Hepia ITI 2ème année

15 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 2 joueurs, 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 "tirer") 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

Le but de mon travail était de mettre en place le graphique du jeu, autrement dit, mettre en place un client qui va se connecter au serveur soit pour lancer une nouvelle partie du jeu, soit pour rejoindre une, créée par un autre joueur.

2.2 Travail réalisé

Pour commencer, j'ai fait quelques recherches sur les frameworks JavaScript qui existent, et que je peux utiliser pour tout ce qui est dessin et manipulation d'objets dans une page web. Finalement, j'ai choisi de travailler avec D3 (Data-Driven Documents v4). D3 est une librairie JavaScript pour manipuler les données et les visualiser utilisant HTML, SVG et CSS.

Dans un premier temps, j'ai commencer par construire deux grilles de jeu qui représentent les cartes de chaque joueur : à gauche, la carte de l'adversaire, et à droite, sa propre carte. Une carte est constituée de 10x10 cellules, donc 100 entiers stockés dans un tableau. Chaque entier représente un identifiant d'une cellule. Ce tableau est passé à une fonction dans D3 pour représenter ses éléments dans une balise SVG de HTML. Après, j'ai construit les bateaux. Ils sont représentés par un tableau d'objets, et aussi passés à D3 pour la représentation graphique.

Dans un deuxième temps, je me suis intéressé à gérer les évenements d'intéraction entre interface et utilisateur (déplacement des bateaux, les clics sur les cases pour tirer, gestion de collision...). Et finalement, j'ai créé une fonction qui permet de placer les bateaux dans la carte d'une façon aléatoire.

Durant une partie, le client communique avec le serveur via différents évènements (expliqué plus en détail dans la section de mon collègue). Par exemple, au moment où le client clique sur "join" pour rejoindre une partie, un évènement spécifique est lancé et envoyé au serveur qui écoute sur celui-là, exécute les traitemets nécessaires et renvoie la réponse au client (changement du contenu de la page HTML).

2.3 (Questions -) Réponses

Qu'est ce qui vous a positivement surpris ?

— L'efficacité de la librairie D3 : après avoir su la syntaxe, la manipulation des objets (insertion, animation, rotation...) devient très facile à coder.

— La facilité de l'échange de données entre le client et le serveur via la librairie socket.io.

Qu'est ce qui vous a pris plus de temps ? La fonction pour placer les bateaux sur la carte d'une façon aléatoire et la gestion de collision entre les bateaux.

Quel est le pire bug que vous avez eu ? Au moment où l'utilisateur clic-droit avec la souris sur un bateau, ce dernier est censé effectuer une rotation de 90 degrés. Par contre, ce qui s'est passé c'est que le rectangle du bateau effectue la rotation, alors que l'image reste dans son état initial. Pour résoudre ce bug, j'ai mis deux images pour chaque bateau : une de direction verticale, et l'autre horizontale. Au moment du clic, je charge l'image entière qui correspond à la nouvelle direction à nouveau.

Au final si vous deviez recommencer que changeriez-vous ? J'implémenterai un moteur graphique pour le jeu, au lieu de travailler avec D3 nativement, pour apprendre de nouvelles techniques.

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 + typage dynamique).

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

Le serveur multi-instance est fonctionnel, le mode joueur contre joueur et joueur contre ordinateur fonctionne, l'interface graphique est propre et fonctionnelle. Il reste un problème qui ne doit pas être très difficile à résoudre : si un joueur (joueur A) en mode joueur contre joueur "join" une partie et attend un autre joueur et qu'un autre joueur (joueur B) rejoint une partie contre l'ordinateur, le joueur A se retrouve à joueur contre l'ordinateur et le joueur B en attente. Pour résoudre cela, il suffirait de changer le mécanisme de "join game", en dédier un unique à l'ordinateur par exemple.

4.2 Propositions d'améliorations

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