Université Grenoble Alpes

M2 MIASHS IC/DCISS

Rapport de projet - TRON

Développement Web Mobile

2022 - 2023

Mohamed Moustafa
Piolat Guillaume
Polydoras Dimitri
Ruffieux Mathis

Sommaire

I) Introduction

Ce projet a été réalisé durant notre 2ème année de Master MIASHS IC/DCISS pour le module de développement web mobile.
L'objectif du projet est de créer une version multijoueur du jeu vidéo Tron qui a pour but de battre son/ses adversaires en les faisant s'écraser sur la traînée qui s'échappe de votre moto.
Le projet a été réalisé en utilisant Cordova, un framework html/css et javascript ainsi que Mongoose pour la base de données.
Dans ce rapport se trouve la répartition du travail entre les membres du groupe ainsi que le détail de l'architecture globale du projet et également nos choix au niveau des structures.

II) Répartition des tâches

Ce projet a été réalisé en groupe de 4, il a donc fallu que nous répartissions les différentes tâches.
Voici comment nous avons travaillé :

Noms	Travail effectué		
	Côté client	Côté serveur	Autre
Guillaume Piolat	- Débogage	- Implémentation BD - Connexion / reconnexion des utilisateurs - Gestion du lobby	- Rédaction rapport - Maquette
Moustafa Mohamed	- Directions du joueur au clavier		- Maquette
Dimitri Polydoras	- Implémentation du jeu - Directions du joueur au clavier		- Rédaction rapport - Maquette
Mathis Ruffieux	- Implémentation du jeu - Implémentation des bots - Refonte et design des vues - Directions du joueur à l'écran - Ajout de la tnt	- Débogage	- Rédaction rapport - Maquette

III) Structure du projet

Le projet est constitué de 3 principaux dossiers :

-Serveur, contenant comme son nom l'indique le code côté serveur
-data, qui contient quant à lui les fichiers liés à la base de données
-Projet_Tron_Application, qui contient le code géré côté client

Pour commencer, l'application est une Single Page Application, ce qui signifie qu'il n'y a qu'une unique page html sur laquelle seront affichées/masquées les différentes composantes avec lesquelles l'utilisateur interagit, ce qui rend les choses plus simples au niveau des requêtes envoyées au serveur et offre une expérience utilisateur optimale, notamment en supprimant l'effet de transition entre les pages.
En outre, elle a été développée sur une base MVC (modèle, vue, contrôleur) à l'exception que nous avons dû séparer les contrôleurs côté serveur des contrôleurs côté clients et qu'il n'existe qu'une seule vue.

Nous allons donc détailler les fichiers de chaque dossier pour expliquer plus précisément le fonctionnement de l'application.

1) Le dossier Serveur

Ce dossier est composé de 5 fichiers javascripts, à savoir :
-Game.js, qui permet de stocker les connexions à chaque joueur d'une partie. Contient aussi les IDs de chaque joueur et surveille l'état courant de la partie (terminée ou non).
-ServerWS.js, qui permet de gérer les Web Sockets (utilisées pour la connexion des clients au serveur) et donc toutes les interactions (messages envoyés) entre les différents clients et le serveur, et agir en conséquence.
-authentication.js, qui s'occupe de l'authentification des clients et de vérifier notamment s'ils ont saisi un mot de passe correct.
-games.js : contient les instances de Game. Cela permet au serveur de gérer plusieurs parties en parallèle.
-lobby.js, qui va gérer la salle d'attente dans laquelle les joueurs patientent avant que la partie ne se lance. Une fois le nombre minimum de joueurs requis atteint, un message sera envoyé à chaque joueur créant leur instance de la partie côté client.

Globalement, ce dossier contient toutes les classes javascript qui nous servent à gérer le côté serveur de l'application et également de relier ceci aux données.

2) Le dossier data

Ce dossier va justement constituer l'essentiel de nos données et de nos accès à la base de données.
Il est composé tout d'abord d'un fichier *seed.js* qui contient des données fictives qui serviront à remplir la base de données pour effectuer des tests.
Ensuite, on trouve un dossier *models* qui contient deux fichiers :
-game.js, qui définit le modèle en BDD d'une partie

-user.js, qui définit le modèle en BDD d'un utilisateur

3) Le dossier Projet_Tron_Application

Ce dossier contient la majeure partie de l'application.
En effet, le jeu étant codé côté client et non serveur, il y a beaucoup plus d'éléments dans ce dossier que dans le dossier Serveur.
Ce dossier contient un répertoire *www* ainsi que différents fichiers de configurations tels que *config.xml*, *package.json*, *package-lock.json*, *.gitignore* et un dossier de configuration *.idea*. On ne s'attardera pas sur ces différents fichiers car ils sont générés par cordova et mongoose.

Au niveau du dossier *www*, on va donc trouver tous les dossiers contenant le code côté client de l'application, répartis en fonction du langage utilisé : *js* contient les fichiers javascripts, *css* l'unique fichier de style de l'application ainsi que *index.html* qui est l'unique vue de l'application, cité précédemment. Il y a également un dossier *img* contenant les images que nous avons intégrées.
Nous allons nous intéresser principalement aux fichiers compris dans le répertoire *js* car il n'y aurait pas grand chose à dire sur la vue *index.html* ou sur la feuille de style lui correspondant.

Le dossier js contient trois fichiers js ainsi qu'un autre dossier :
-event_listeners.js, qui est un fichier qui va gérer les actions liées aux clics sur les différents boutons présents dans l'application
-index.js, qui va permettre de gérer l'envoi et la réception des messages durant la communication avec le serveur. Ce fichier est très important car il permet notamment de notifier le serveur que le joueur veut tourner sa moto ou encore de gérer l'affichage du nouvel état de la partie pour le joueur
-ui.js, qui va simplement définir toutes les fonctions nécessaires à l'affichage des modules composant la vue de l'utilisateur
Et enfin jeu, qui est un autre dossier dont nous allons détailler le contenu.

Ce dossier contient les éléments en rapport avec le jeu en lui-même, côté client.
Il contient donc deux fichiers javascript et un dernier autre dossier :
-dom.js, qui contient les fonctions liées aux éléments du DOM, notamment rafraîchir le terrain avec les différentes couleurs en fonction de leur valeur html qui est modifiée lors du jeu
-jeu.js, qui va contenir quant à lui toutes les variables liées au jeu ainsi que les fonctions qui vont permettre au jeu de fonctionner telles que l'initialisation, faire avancer un joueur, le faire tourner en fonction de la touche pressée et vérifier l'état de la partie, avant d'envoyer les messages contenant toutes ces informations au serveur ainsi que toutes les fonctions qui gèrent les mouvements des bots.
Et pour finir, le dossier Models.
Ce dossier contient les fichiers suivants :
-Block.js, qui va définir la classe qui correspond à chaque carreau de notre grille de jeu et qui va définir ses caractéristiques à savoir principalement sa couleur, s'il s'agit d'un mur ou non et s'il y a un joueur dessus ou non

-Player.js, qui définit la classe liée à un joueur ainsi que tous les attributs qui vont le constituer, la fonction qui vérifie si le joueur s'est pris un mur ou non ainsi que la fonction qui code le déplacement des bots de manière plus ou moins aléatoire
-Terrain.js qui va permettre de construire le terrain jouable en initialisant la grille de Blocks, et en ajoutant les joueurs dessus

4) Fonctionnement du jeu

Pour parler du jeu en lui-même, nous avons décidé d'implémenter une version jouable à 4 joueurs. Sachant qu'il n'est souvent pas chose aisée de se retrouver à 4 pour jouer et étant conscients de la difficulté supplémentaire qu'aurait été la mise en place du choix du nombre de joueurs, nous avons décidé de fixer la taille à 4 joueurs et d'ajouter des bots pour compléter les places manquantes dans la partie, la partie n'étant cependant jouable qu'à partir du moment ou il y a au minimum 2 joueurs.

5) Fonctionnement des bots

Lorsque 2 joueurs sont dans le lobby, la partie démarre et le joueur 1 héberge les bots sur son client. Ainsi, lorsque les bots effectuent des mouvements, ils envoient au serveur un message pour que chaque client dispose des mêmes mouvements de bots.

Le fonctionnement d'un bot est le suivant :
Il avance tout droit et à une petite chance de tourner vers sa gauche ou vers sa droite. Lorsqu'il se trouve à 2 cases ou moins d'un mur dans sa direction, il tourne aléatoirement sur une case sans obstacle. Ainsi les bots ne rentrent dans un mur que lorsqu'ils sont dans un cul de sac.

6) Le gameplay

Pour les flèches directionnelles, l'utilisateur a la possibilité d'utiliser son clavier, ou de cliquer sur les flèches à l'écran. Lorsque l'utilisateur change de direction, une information visuelle est affichée sur les flèches afin qu'il comprennent facilement sa direction.
De même, une information est affichée pour lui dire explicitement sa couleur et ainsi s'y retrouver lorsque la partie commence.

Nous avons décidé d'implémenter le même schéma sur téléphone en proposant des flèches directionnelles également, cliquables cette fois ci de manière tactile.

7) Répartition coté client et serveur

Nous avons décidé de mettre le maximum d'éléments côté client, le serveur servant simplement de passe-plat, pour deux raisons :
Premièrement, nous sommes beaucoup plus à l'aise avec le code côté client : en effet, la majorité des applications que nous avons développé auparavant étaient des applications essentiellement côté client, ce qui nous a permis d'être plus efficaces durant les phases de développement.
Deuxièmement, il est moins coûteux lors d'un déploiement à grande échelle de faire fonctionner le jeu côté client, cela permet d'avoir moins de données à traiter côté serveur et donc d'assurer une meilleure stabilité.

En cas de changement de direction d'un joueur, le serveur transmet simplement aux autres joueurs de la partie l'information que tel joueur a changé de direction, les calculs étant faits côté client. Celà fonctionne aussi pour les bots.

Si un joueur se déconnecte en pleine partie et se reconnecte à nouveau, le serveur demande à un autre joueur de la partie l'état de son jeu. Le serveur transmet alors cet état de jeu au client qui a été déconnecté.

8) Bonus : la tnt

De la même manière que les bots, la génération aléatoire de tnt fonctionne sur le client du joueur 1 puis envoie un message au serveur pour notifier tous les clients. Ainsi chaque client aura le même plateau.
Lorsqu'un joueur passe sur une tnt, celle-ci explose détruisant les murs générés par les joueurs se trouvant jusqu'à 20 cases autour d'elle.
Les bots tournent sur la tnt lorsqu'ils passent à côté pour l'actionner et se libérer d'éventuels cul de sac.

La tnt ne tue pas les joueurs, en effet ce n'est pas un jeu de bomberman. Cette fonctionnalité permet d'allonger le temps d'une partie en détruisant des blocs et ajouter un côté plus dynamique au jeu.

9) Arborescence de l'application

Pour avoir une idée plus précise de la structure du code, voici un arbre représentant les différents composants de notre application :

Projet_Tron ▽
    Projet_Tron_Application ▽ -Code côté client
        .gitignore

config.xml
package-lock.json
package.json
.idea ▷
www ▽
- *index.html* -unique fichier html (SPA)
- css ▽
  - *index.css* -unique fichier css, définit le style
- img ▷ -images utilisées
- js ▽
  - *event_listeners.js* -gère les clics des boutons
  - *index.js* -communication client/serveur
  - *ui.js* -gère l'affichage sur la vue
  - jeu ▽
    - *dom.js* -gère les éléments du DOM
    - *jeu.js* -fonctions utiles au fonctionnement du jeu
    - models ▽ -définissent les modèles utilisés dans le jeu
      - *Block.js*
      - *Game.js*
      - *Player.js*
      - *Terrain.js*

Serveur ▽ -Code côté serveur
- *Game.js* -partie
- *ServerWS.js* -communication client/serveur
- *authentication.js* -authentification
- *games.js* -contient les parties
- *lobby.js* -salle d'attente

data ▽ -Gestion BDD
- models ▽
  - *game.js* -modèle BDD d'une partie
  - *user.js* -modèle BDD d'un utilisateur
- *seed.js* -données de test