TETONOR
Un jeu web interactif multijoueur

Cláudio Sousa

Décembre 2018

Table des matières

1 Introduction

Ce projet consiste à développer un jeu multijoueur basé sur des technologies web mettant en pratique la théorie du cours de Technologies Web avancées suivit à l'HEPIA [1] pendant le semestre d'automne 2018 avec le professeur Stéphane Malandain.

Ce jeu permet à plusieurs joueurs de jouer une même partie de Tetonor de manière simultanée et concurrente. Chaque joueur joue sur son navigateur web, avec l'objectif d'être le premier à terminer le jeu.

Dès qu'un joueur termine le jeu, le jeu est fini et le joueur est déclaré vainqueur.

2 Code source

Le projet est disponible en open-source sur Github à l'url `https://github.com/claudiosousa/tetonor`.

2.1 Installation

L'application utilise Node.js [2] et il doit être installé sur le système pour pouvoir tourner l'application.

Pour installer l'application, la première étape est de cloner le dépôt Git :

```
git clone git@github.com:claudiosousa/tetonor.git
```

Ensuite, il est nécessaire d'installer localement les dépendances du projet :

```
npm install
```

Il ne reste alors qu'à démarrer l'application :

```
npm start
```

L'instruction ci-dessus va également ouvrir une page de votre navigateur sur l'url de l'application `http://localhost:7654/`.

Pour plus d'informations, lire le *readme* du projet sur Github.

1. Haute école du paysage, d'ingénierie et d'architecture de Genève (*https://www.hesge.ch/hepia/*)
2. *https://nodejs.org/*

2.2 Structure du code source

Voici les fichiers et dossiers les plus importants sur Github :

```
report ..................................................dossier content ce rapport
package.json ..................................instructions pour les packages npm
src ..........................................................racine du code source
  server ..................................................... le serveur web
    CommunicationManager.js ..................service gérant les communications
    Game.js ..............................................l'état d'un jeu en cours
    GameBoard.js ..............................................la logique du jeu
    index.js ..............point d'entrée avec initialisation web et websocket
  webapp ..................................................l'application web
    components ..................................................composants UI
      choose-game.js ..........................composant UI de choix de partie
      join-game.js ..............composant UI permettant de joindre une partie
      waiting-players.js ............composant UI d'attente des autres joueurs
      tetonor.js.........composant UI pour joueur le jeu Tetonor .4 problem.js
        composant UI pour une case du jeu
      tetonor-input.js ..................................un input pour une case
      player-score.js .............composant UI affichent le score des joueurs
      game-over.js ................................composant UI de fin de partie
    services ..................................................services applicatifs
      communication-service.js ................service gérant la communication
      game-manager.js ..........................service gérant le jeu en cours
    style ..................................................css et images
    app.js ..................................... création de la vue principale
    index.html ..................................................la page HTML
```

3 Déroulement d'une partie

3.1 Joindre une partie

Les utilisateurs jouent des *parties* ensemble. Une partie est constituée d'un ensemble de joueurs qui essaient simultanément de résoudre le même jeu de Tetonor.

L'utilisateur peut joindre une partie déjà existante en choisissant une parmi celles qui sont listés. L'image 1 montre la liste de parties créés. Leur nom est affiché à gauche, et à droite il est affiché le nombre de joueurs ayant déjà rejoint la partie ainsi que le nombre minimal de joueurs nécessaires pour commencer la partie. Les parties commencées sont affichées en bleu, et les autres sont en vert.

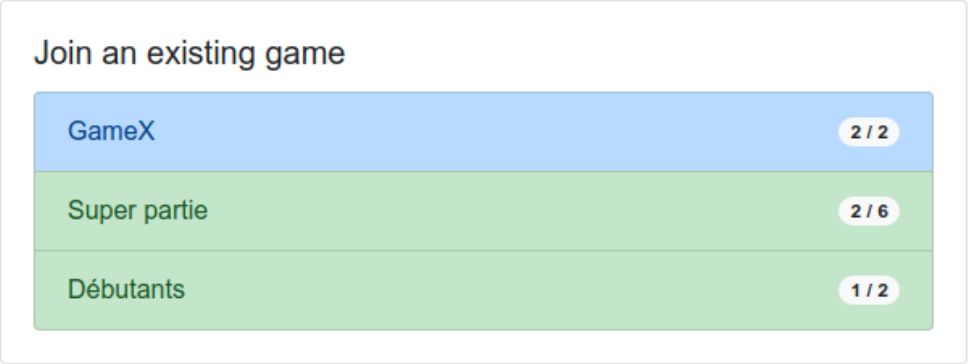

FIGURE 1 – Liste des parties existantes

L'utilisateur peut aussi choisir de créer une nouvelle partie, en spécifiant un nom et un nombre minimal de joueurs, comme le montre l'image 2.

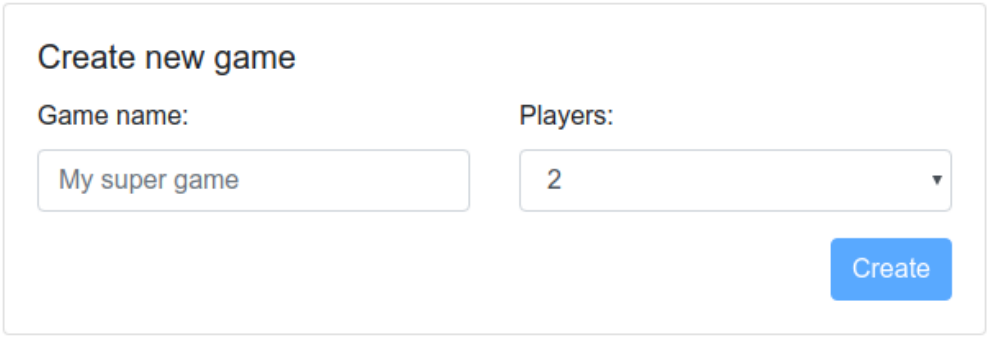

FIGURE 2 – Création d'une nouvelle partie

Les joueurs qui rejoignent une partie sont mis en attente jusqu'à ce que le nombre minimal de joueurs aie rejoint la partie. En attendant, le nombre de joueurs ayant rejoint la partie est affiché en continu comme le montre l'image 3.

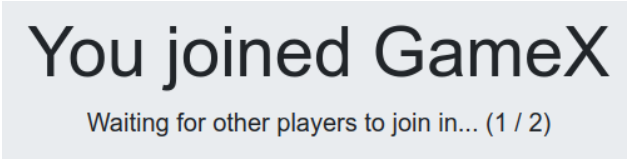

FIGURE 3 – Attente des autres joueurs

Dès que le nombre de joueurs minimal est présent, la partie commence. À noter qu'il n'y a pas de nombre maximal de joueurs. Il est toujours possible à de nouveaux joueurs de joindre une partie qui a déjà commencé.

3.2 Le jeu

Une fois la partie commencée, chaque joueur reçoit du serveur le même jeu de Tetonor à résoudre.

Le jeu se joue en déplaçant les numéros disponibles sur les cases à remplir. L'opérateur se choisit en cliquant sur la case entre les numéros.

L'image 5 montre un exemple d'une partie en cours. Les numéros corrects sont affichés en bleu et les incorrects en rouge.

Sur la colonne de droite, la liste des joueurs en cours est affichée avec leur taux de progression.

Le joueur peut ainsi mesurer son progrès relativement à ses adversaires.

Les joueurs dont le nom apparaît en gris sont des joueurs ayant quitté la partie.

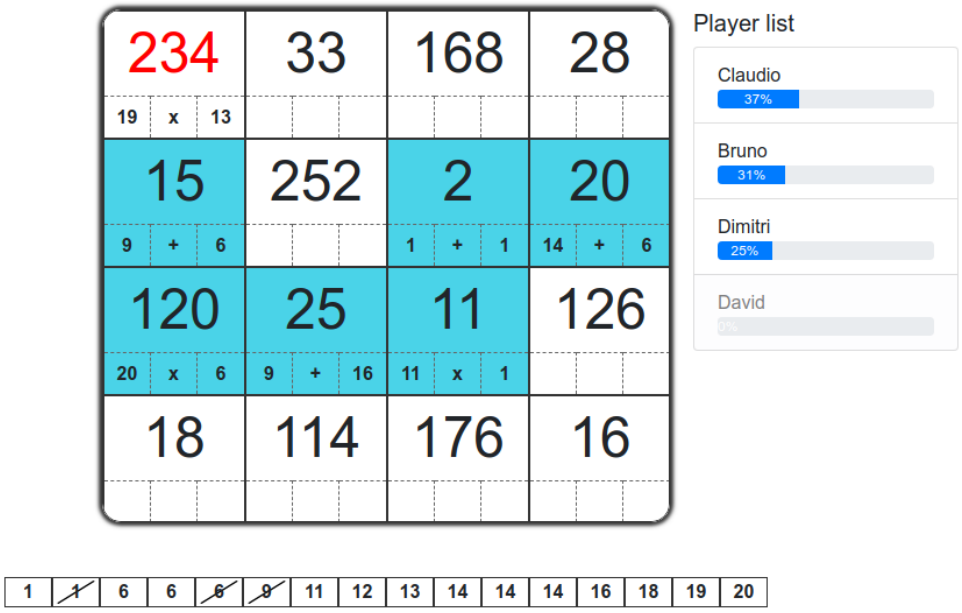

FIGURE 4 – Une partie en cours

Le premier joueur ayant complété le jeu à 100% gagne la partie.

Tous les joueurs sont alors informés que la partie est finie et leur résultat est affiché (*gagné* ou *perdu*).

3.3 Échanges entre client web et le serveur

Le diagramme ci-dessous, montre les principales échanges entre le client web et le serveur :

client-server.png client-server.png

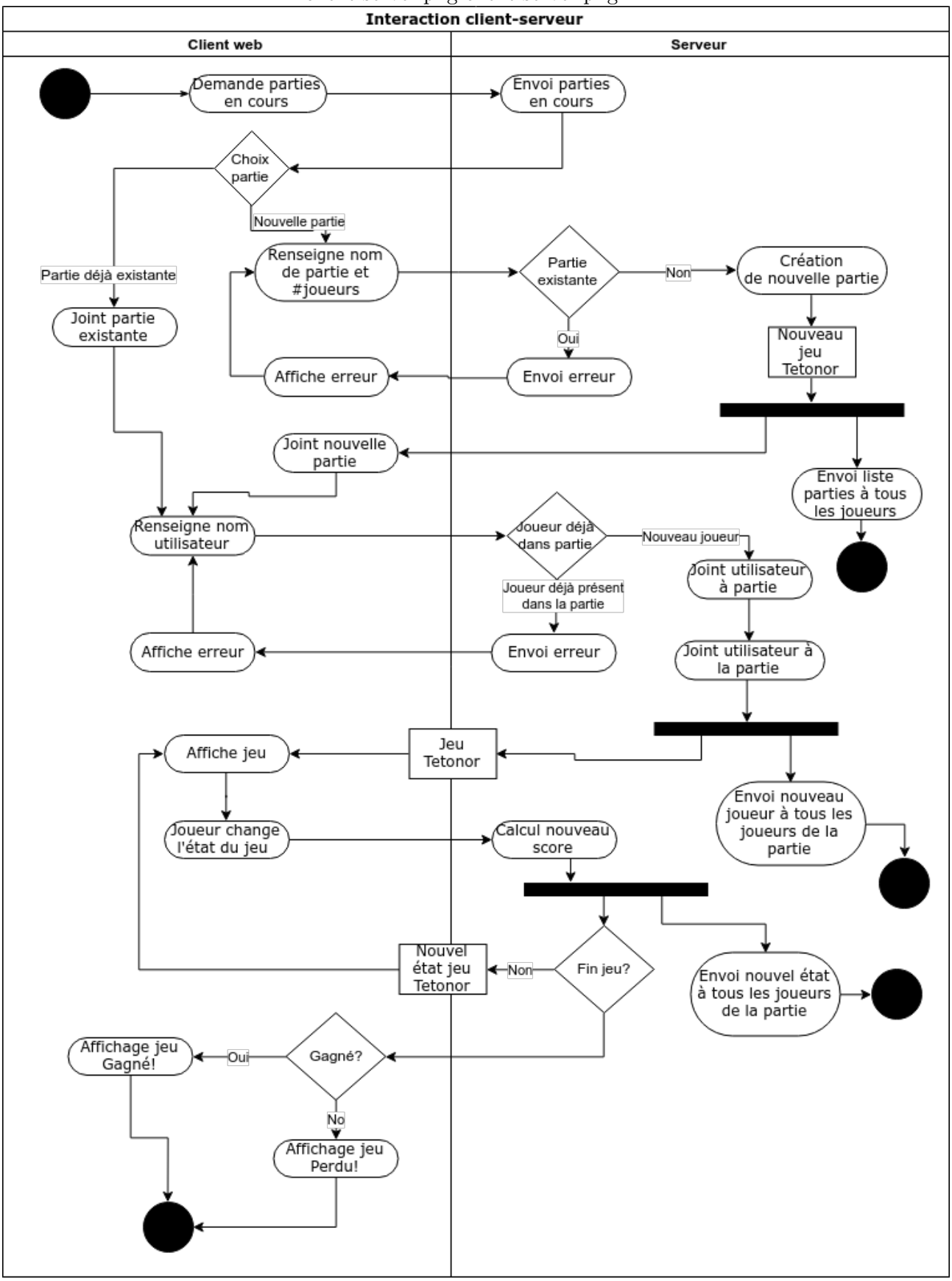

FIGURE 5 – Une partie en cours

4 Architecture

L'architecture est composée d'une application web et d'un serveur web.

La communication entre les clients web et le serveur se fait en utilisant des requêtes Ajax [3] ainsi qu'en utilisant des WebSockets [4]. Ces dernières permettent une connexion bidirectionnelle avec le serveur. Le serveur les utilise pour envoyer (*push*) des informations sur les clients.

Aucune connexion n'a lieu entre deux clients web directement.

4.1 Application web

L'application web est une application mono-page, réalisée en utilisant les libraires Vue.js [5] et Boostrap [6].

À noter que le pourcentage de progression montré ainsi que la solution pour une partie en cours n'est pas décidée par le client mais par le serveur. Le serveur suppose que le client peut être compromis et comme tel ne lui fait pas confiance. À chaque fois que l'utilisateur fait une opération sur la partie en cours, l'état de son jeu est envoyé au serveur. Ce dernier va comparer ce que l'utilisateur envoie avec l'état du jeu généré par le serveur pour la partie en cours.

Pour déterminer le nouveau score de la solution envoyée par le client, le serveur vérifie que les choix de numéros et opérateurs faits par l'utilisateur permettent de résoudre les cases du problème. Le serveur vérifie aussi que les numéros utilisés dans la solution sont ceux générés pour la partie concernée et que le client n'utilise que la quantité qu'il a disponible.

4.2 Le serveur

Le serveur rempli 2 rôles :

— serveur web, afin de servir les fichiers de l'application web
— serveur applicatif supportant la logique du jeu

Il utilise principalement les libraires Express [7] et Express-ws [8] comme support de connexion HTTP et WebSockets.

5 Conclusion

Ce projet a été pour moi l'opportunité de résoudre des problèmes dans les domaines de l'architecture, l'ingénierie logicielle, l'expérience utilisateur et le design.

3. *https://en.wikipedia.org/wiki/Ajax_(programming)*
4. *https://en.wikipedia.org/wiki/WebSocket*
5. Framework web utilisé pour construire des interfaces utilisateur pour des applications mono-page (*https://vuejs.org/*)
6. Collection de composants graphiques web (*https://getbootstrap.com/*)
7. Librairie offrant des capacités de serveur http (*https://expressjs.com/*)
8. Librairie supportant un serveur websockets (*https://www.npmjs.com/package/express-ws*)

Sur le plan **architectural**, j'ai pu créer une architecture qui permet au serveur de gérer plusieurs clients connectés simultanément, travaillant ensemble de manière structurée et cohérente. Les mécanismes de communication permettent au serveur de recevoir et d'envoyer de messages aux clients individuellement, mais aussi d'envoyer des messages à tous les clients ou à un sous-ensemble.

Sur les aspects **algorithmiques**, il a fallu concevoir des composants permettant la gestion de plusieurs parties ayant lieu simultanément. Chaque partie à sa propre liste de joueurs et sa propre machine à états. Une deuxième difficulté de ce chapitre fut pour moi la définition des événements et messages nécessaires à la synchronisation des différents clients jouant une même partie.

Au sujet de l'**expérience utilisateur**, la problématique des parties multiples a été la plus difficile à présenter clairement à l'utilisateur. La solution retenue qui présente la liste des parties existantes tout en permettant d'en créer des nouvelles est une solution que je crois satisfaisante.

Je suis également satisfait du composant qui montre à l'utilisateur le progrès des différents joueurs de la partie. Cette information rend le jeu plus compétitif et donc plus intéressant.

Concernant le **design**, il a fallu faire des efforts pour rendre l'application web esthétiquement agréable. L'esthétique est un aspect qui passe souvent après la fonctionnalité, mais qui ne doit pas être négligé.

Au sujet des technologies utilisées, ce projet fut pour moi l'opportunité d'apprendre à utiliser Vue.js. C'est un concurrent sérieux de React.js et Angular auquel je m'intéressai depuis un moment. Je suis très content d'avoir pu compléter la connaissance apprise en cours sur Angular avec celle apprise en pratique sur Vue.js.

Au sujet de la charge de travail, j'ai sous-estimé l'effort nécessaire pour amener le projet à terme. A posteriori, il est clair pour moi que le projet était suffisamment grand pour être partagé et il aurait été plus judicieux de le faire à deux.

Pour résumer, ce projet a été à la fois intéressant et ludique grâce à la complémentarité de la théorie reçue en cours et la mise en pratique de cette dernière.