Université Grenoble Alpes

10/01/2025

Projet - Jeu de Dames

Rapport - Développement Web Mobile

Di Malta Gaël
Espinal Miguelangel
Genevey Coemgen
Philippe Aurore

Sommaire

1. Introduction

Ce projet s'inscrit dans le cadre du cours de développement Web Mobile, dispensé en deuxième année de master MIASHS, parcours IC, à l'université Grenoble Alpes. Il doit mener à la conception d'un jeu de dames couplé avec un serveur et au moins deux clients pour permettre aux étudiants de se familiariser avec les technologies vues en cours. Pour ce faire, l'application doit être une single page application, destinée aux joueurs de dames, qui doit permettre à un utilisateur de s'identifier, attendre pour un deuxième joueur de se connecter, jouer aux dames contre un autre utilisateur, finaliser le match et de consulter les statistiques (nombres de parties jouées, gagnées et perdues) de tous les joueurs y compris de lui-même. La date du rendu final du projet est fixée au 10 janvier 2025.

Parties prenantes

Le projet mobilise les acteurs suivants :

Rôles	NOM Prénom
Enseignant et coordinateur pédagogique	VAUFREYDAZ Dominique
Membre de l'équipe projet	DI MALTA Gaël
Membre de l'équipe projet	ESPINAL Miguelangel
Membre de l'équipe projet	GENEVEY Coegmen
Membre de l'équipe projet	PHILIPPE Aurore

Répartition du travail

Nous nous sommes répartis le travail comme suit :

Tâches	NOMS Prénoms
Côté serveur (Node.js) et base de données (MongoDB)	DI MALTA Gaël ESPINAL Miguelangel
Côté client (Cordova, HTML, CSS, JavaScript)	GENEVEY Coegmen PHILIPPE Aurore

2. Exigences fonctionnelles

Fonctionnalités et services

L'application proposera les services suivants :

- Connexion (et inscription) à l'application à l'aide d'un pseudo et d'un mot de passe.
- Lancer et jouer une partie de dames en 1VS1.
- Consulter les statistiques de tous les joueurs.

Plateformes

L'application devra être disponible sur Mobile et sur navigateur Web lancé dans un environnement local.

Technologies

Le développement de l'application devra se faire avec Cordova, Node.js et MongoDB.

Outils

Nous utilisons GitHub avec trois dépôts : un pour le serveur, un pour le client et un qui lie les deux autres dépôts. De plus, nous utilisons Google Drive pour la réalisation de nos rapport et Discord pour s'organiser et communiquer entre nous.

Liaison des git : https://github.com/migesve/Jeu-dames-EDGP
Côté client : https://github.com/Calu12/JeuDame-Espinal-Genevey-Philippe-DiMalta.git
Côté serveur : https://github.com/Cybermiam/JeuDameServer.git

3. Solution

Architecture

L'architecture de l'application repose sur une séparation claire entre le client et le serveur. Le serveur gère la connexion des utilisateurs, la mise en file d'attente, le lancement des parties, ainsi que la gestion des données dans une base dédiée. Les messages échangés entre le client et le serveur utilisent un format JSON standardisé. Côté client, l'interface offre une navigation simple entre les sections principales : connexion, tableau des scores et partie de jeu, avec un plateau interactif conçu en SVG pour une meilleure expérience utilisateur. Les détails suivent ci-dessous.

Côté serveur

Pour la partie serveur de l'application, les choix suivants ont été faits :

- Gestion de la connexion
    - Un utilisateur se connectant pour la première fois devra créer un compte qui sera stocké dans la base des données.
    - Les autres devront se connecter, et suite à l'interrogation de la base des données, si les informations sont correctes, on met l'utilisateur en file d'attente.
- File d'attente et lancement d'une partie
    - Chaque utilisateur est mis en file d'attente après la connexion.
    - On attend un deuxième joueur disponible.
    - Dès que possible, on lance la partie avec un autre joueur disponible.
- Base de données
    - La base de données doit pouvoir conserver des utilisateurs, les consulter, et sauvegarder des résultats de parties.
- Les message de l'application pour communiquer client - serveur sont des messages JSON sous la forme suivante :

```
{
        type :  X,
        attribut1 : Y,
        attribut2 : Z,
        attribut[i] : …
}
```

    - EXEMPLE :

      Message de connexion

```
{
        type: "login",
        username: "username",
        password: "password"
}
```

      Message de mouvement

```
{
        type: "move",
        username: "usernameDuJoueurQuiBouge",
        positionDepart: [x, y]
        positionArrive: [newX, newY]
}
```

Le serveur redirige le message à la bonne API en fonction de la variable type.
Les messages à implémenter sont :

- Message de login (accepter, refuser la connexion, erreur dans les coordonnées).
- Message entre les deux clients pour les mouvements des pièces, en passant par le serveur.
- Message distribué par le serveur aux clients pour les conditions de victoire.

- Gestion des déconnexions :
    - Dans un premier temps, nous considérerons qu'une déconnexion en cours de partie est équivalente à un abandon, ce qui attribuera la victoire à l'adversaire. Si le projet progresse bien, nous pourrions envisager la mise en place d'un délai déterminé : si le joueur se reconnecte avant la fin de ce délai, il pourra reprendre le match, sinon, cela sera considéré comme un abandon.

Côté client

Pour la partie client de l'application, nous avons décidé de masquer ou d'afficher les sections suivantes du code : la section liée à la connexion des utilisateurs, celle liée à l'affichage des scores ainsi qu'au lancement d'une partie de jeu, et celle liée à une partie de jeu en cours. De plus, nous avons décidé de concevoir le plateau de jeu en utilisant le format SVG pour permettre un développement ainsi qu'une évolutivité du plateau plus simple et rapide améliorant de cette façon l'expérience utilisateur plus facilement. De plus, nous sommes déjà familier avec SVG.

Pour une navigation claire et simple entre les sections, notre application prendrait forme comme suit:
Lors du lancement de l'application, un utilisateur est redirigé vers un écran de connexion, où il peut saisir un pseudo et un mot de passe. Une fois rentrées, ces informations de connexion sont ensuite traitées par le serveur. Une fois la connexion validée, l'utilisateur accède à une page affichant un tableau des scores, qui indique le nombre de parties gagnées par joueur, ainsi qu'un bouton permettant de lancer une partie de jeu.
En cliquant sur le bouton qui permet de lancer une partie, l'utilisateur entre dans une file d'attente où il attend un adversaire. Une fois un joueur disponible, une partie commence. Lorsqu'une partie prend fin, soit par son achèvement, soit par déconnexion de l'un des joueurs, les joueurs sont redirigés vers la page des scores.
Un utilisateur peut se déconnecter à tout moment depuis les pages des scores ou de jeu. Une fois déconnecté, il est automatiquement renvoyé vers la page de connexion.

Le client peut envoyer les message suivant au serveur :

- demande de connection :

{
type: 'login',
username: “nom de l’utilisateur”,
password: “mot de passe de l’utilisateur”
}

- demande de déconnexion :
  {
  type: 'logout',
  username: ''nom de l’utilisateur”
  }

- recherche de partie :
  {
  type: 'startGame',
  username: “nom de l’utilisateur”
  }

- demande de déplacement :
  {
  type: 'move',
  username: “nom de l’utilisateur”,
  positionDepart: [x, y]
  positionArrive: [newX, newY]
  }

Quand la demande de déplacement est envoyée au serveur, on attend la réponse du serveur pour savoir si l’action est possible, auquel cas on l’exécute sur l’image du plateau.

4. Application finale

Implémentations

Les implémentations en fin de projet permettent aux clients de se connecter ou de s'inscrire, de lancer une partie avec un adversaire de la file d'attente et de déplacer les pions simples de sa couleur quand c'est à son tour de jouer.

La base de données de la solution présentée précédemment a bien été implémentée dans les délais du projet impartis. De plus, le serveur gère la connexion, la file d'attente et le lancement d'une partie comme cela a été annoncé.

Un client peut naviguer entre les pages de connexion, de lancement d'une partie de jeu et d'une partie en cours. De plus, nous avons pu implémenter les règles de déplacement et de capture d'un pion côté client.

Difficultés rencontrées

Nous n'avons pas pu implémenter toute la solution voulue, notamment à cause des difficultés que nous avons rencontré durant le projet. En effet, nous avons, notamment, eu des difficultés à faire correspondre la grille svg, affichée côté client, avec la grille stockée dans le serveur lors des déplacements de pions de dame.

Finalement, nous n'avons pas eu le temps de gérer la déconnexion des clients ainsi que l'affichage du tableau des high scores. Un client ne peut donc pas se déconnecter ni consulter les statistiques de tous les joueurs. De plus, nous n'avons pas gérer les dames par manque de temps.