Master MIASHS 2
Université Grenoble Alpes
Décembre 2025

Application Web/Mobile

Rapport de projet

Groupe 2
DESCHAMPS Kylian
DIONE Ayla
JACQUET Léo
LUGINBUHL Valentin
AROLD ROSEMOND Marc

Table des matières

Introduction et présentation du projet

Le projet a été réalisé dans le cadre du Master 2 MIASHS à l'Université Grenoble Alpes. Notre objectif principal était d'explorer et de maîtriser deux enjeux fondamentaux du développement d'applications modernes : la communication temps réel via le protocole WebSocket et la gestion du déploiement multiplateforme via une application mobile hybride.

Gestion de projet

Avant de développer tête baissée, nous avons planifier l'architecture et les étapes de développement (voir document planification.pdf).

Structure du projet

Le projet repose sur la structure générale suivante:

- Serveur : Node.js avec le module websocket pour la logique métier et la diffusion de l'état.
- Client Web : HTML5, CSS3, et JavaScript pour une Single Page Application réactive.
- Mobile : Apache Cordova pour l'encapsulation et le déploiement multiplateforme (Android/iOS).
- Base de données : MongoDB avec Mongoose pour la persistance des comptes utilisateurs et de l'historique des parties.

Architecture générale

Le projet repose sur une architecture avec communication bidirectionnelle entre un serveur et des clients via WebSocket. Cette approche garantit une synchronisation en temps réel, indispensable pour un jeu multijoueur. Le serveur est implémenté en Node.js et gère la logique de jeu ainsi que la persistance des données dans MongoDB via Mongoose. Le client web est construit en HTML5, CSS3 et JavaScript et s'exécute dans le navigateur.

L'application mobile utilise Apache Cordova pour un déploiement multiplateforme. L'ensemble partagent le même protocole de communication et le même état de jeu.

Flux de données

Le cycle de vie d'une partie suit un enchaînement simple et déterministe. Les clients envoient des requêtes au serveur pour s'authentifier, rejoindre une file d'attente ou modifier leur direction. Le serveur orchestre la mise en relation des joueurs, démarre les parties lorsque les lobbies sont complets, exécute la boucle de jeu, calcule mouvements et collisions, puis diffuse l'état courant vers l'ensemble des clients connectés. La base de données, elle, sert au stockage des comptes et de l'historique des parties terminées. Ce découplage permet au serveur d'être réactif sur le temps réel tout en conservant la mémoire à long terme nécessaire au classement global.

Choix architecturaux fondamentaux

Nous avons finalement adopté une architecture où le serveur exécute le jeu. Lors des premières réflexions, l'idée que chaque client calcule la partie localement semble logique pour la charge du serveur. Toutefois, les difficultés de synchronisation, de cohérence et de robustesse face aux déconnexions rendaient cette piste fragile. En multijoueur temps réel, de légères dérives d'horloge, des variations de latence ou un ordre de traitement divergent suffisent à produire des états contradictoires. En centralisant les calculs, le serveur devient la source de vérité, arbitre les collisions, diffuse l'état à intervalle régulier et simplifie la gestion en cas d'incident avec un client. Les clients se concentrent sur l'interface et l'envoi des directions, ce qui améliore la stabilité et réduit les possibilités de triche.

Structure de la base de données

La base MongoDB comporte deux collections principales: « users » pour l'authentification et « games » pour l'historique des parties. Les comptes sont identifiés par un nom d'utilisateur unique et un mot de passe haché, avec des dates de création et de mise à jour. Lors d'une tentative de connexion, si l'utilisateur n'existe pas, il est créé automatiquement, ce qui simplifie l'expérience pour ce projet. Les parties sauvegardent la liste des joueurs, leur score et leur rang final du premier au dernier. Nous conservons uniquement les données nécessaires au calcul du classement et à l'affichage d'un historique, les informations de gameplay détaillées (positions, couleurs, trajectoires) restent éphémères et ne sont utiles que pendant la partie. Ce design minimaliste s'inscrit dans la logique NoSQL où on retrouve de la redondance dans la base de données et sans appel à des jointures extérieures pour gagner en rapidité grâce au NoSQL sur la lecture de la base.

Format des communications

Tous les échanges client serveur se font en JSON via WebSocket. Chaque message suit une convention commune avec un attribut « type » qui identifie l'action ou la réponse correspondante, et, côté serveur, un indicateur « ok » qui précise la réussite de l'opération, éventuellement accompagné d'un message d'erreur explicite. Cette standardisation facilite le routage côté serveur et simplifie le débogage côté client.
Voici les différents types de paquets envoyés entre le serveur et les clients :

- Messages client : L'authentification (« login »), la demande de rejoindre une partie (« joinGame »), le retrait de la file (« leaveQueue »), le changement de direction (« changeDirection ») et la requête de classement global (« getDashboard »).
- Messages serveur : L'état complet de la partie à chaque tick (état global, détail de chaque joueur, etc.) et le classement des joueurs

agrégé sur demande.

Chaque paquet reçu est validé avant traitement (présence/cohérence des paramètres, existence des entités, compatibilité de l'action) pour protéger le serveur des incohérences client, des effets de latence et des comportements inattendus.

Côté client, plusieurs optimisations limitent le trafic inutile (non envoi des commandes de direction identiques, filtrage des demi-tours interdits, désactivation des boutons après action).

Fonctionnalités de l'application

L'application propose les fonctionnalités suivantes :

- Parties multijoueurs de deux à quatre joueurs.
- Authentification unifiée (connexion/inscription).
- Files d'attente par mode de jeu et parties simultanées indépendantes.
- Tableau de classement basé sur les victoires.
- Interface responsive utilisable sur web et mobile.
- Système de couleurs aléatoires sans doublon.
- Gestion de la déconnexion d'un joueur.
- Système de compte à rebours pour lancer une partie.
- Gestionnaire des événements de touche enfoncée.
- Contrôle mobile.

Logique de jeu

La grille de jeu comporte 50×50 cases (paramètre configurable). Les positions de départ sont placées dans les coins pour maximiser la symétrie. Les couleurs sont attribuées aléatoirement dans un ensemble restreint sans doublons afin d'assurer une lisibilité immédiate.

La boucle s'exécute à une fréquence de dix ticks par seconde. À chaque tick, le serveur déplace tous les joueurs vivants et prolonge leur traînée, puis évalue les collisions (sortie de grille, traînée personnelle, traînée adverse). Les nouvelles positions et les collisions sont calculées globalement avant d'appliquer les changements d'état. Cette simultanéité garantit que deux joueurs qui se percutent au même instant meurent réellement en même temps. Après chaque tick, l'état est diffusé aux clients pour un affichage synchronisé

Le score final combine trois dimensions complémentaires :

1. Progression : Récompensée par la longueur de la traînée.
2. Éliminations : Un bonus partagé est attribué aux survivants.
3. Rang : Un bonus de rang appliqué en fin de partie, adapté au mode de jeu.

La déconnexion en cours de partie est assimilée à une mort immédiate. L'avatar cesse de bouger et sa traînée reste sur le plateau comme obstacle permanent.

Rendu graphique

Nous avons privilégié le rendu SVG plutôt que en pixels ou Canvas pour bénéficier d'une scalabilité vectorielle sans perte de qualité, ce qui simplifie considérablement l'adaptation de l'interface à des écrans de tailles très différentes (du smartphone à l'écran d'ordinateur) tout en garantissant une netteté parfaite et une manipulation DOM aisée côté JavaScript. Par dessus la grille du jeu nous avons ajouté des assets pour rendre le jeu plus accueillant et attrayant.
Crédits : https://pixelfrog-assets.itch.io/tiny-swords

La répartition des tâches a été la suivante :

- DESCHAMPS Kylian : Documentation/Rapport

- DIONE Ayla : Planification/Cahier des charges
- JACQUET Léo : Server/Fullstack
- LUGINBUHL Valentin : Front/Déploiement
- AROLD ROSEMOND Marc : Planification/Cahier des charges

Déploiement

Nous avons décidé de déployer le projet sur un serveur pour faciliter la connexion au serveur. Le serveur ainsi que l'instance mongodb sont contenus dans un container docker (voir dockerfile) sur un serveur distant derrière un reverse proxy pour pouvoir activer le protocole https avec le client. Les fichiers client sont distribués depuis sur un serveur web.

Serveur : https://sheep-e-eat-project.valentinluginbuhl.fr/
Client : https://valentinluginbuhl.fr/sheep-e-eat-project/

Conclusion

Ce projet nous a permis d'aborder concrètement le développement web d'un jeu. Nous avons été obligés de passer par des phases de réflexion pour considérer les choix possibles et leurs conséquences.Ce fut notamment le cas pour l'arbitrage entre une logique centrée sur le serveur ou sur le client, un choix déterminant pour la gestion des événements déterminant la communication entre le serveur et les clients.

Dans la conception, la création du jeu nous a amenés à réfléchir à l'équilibrage des mécaniques comme la définition des règles d'égalité, le positionnement des joueurs, mais aussi à la gestion de cas spécifiques mais pouvant entraîner des erreurs, tels que la déconnexion d'un joueur.

L'implémentation et l'utilisation de Cordova pour le passage sur mobile force également la réflexion sur comment désigner le jeu, réfléchir aux

déplacements mais plus généralement prendre en compte la logique d’utilisation d’un téléphone qui diffère d’un ordinateur avec un clavier et une souris.

L'intégration de MongoDB nous a permis d'exploiter la flexibilité du schéma NoSQL face aux besoins évolutifs du jeu, et permettant d’investir un temps minimum sur celle-ci.