Chess Games
Rapport de soutenance

Matthis Guillet
matthis.guillet@epita.fr

Martin Madier
martin.madier@epita.fr

Martin Pasquier
martin.pasquier@epita.fr

Matteo Wermert
matteo.wermert@epita.fr

M&Chess

EPITA

25 février 2025

Table des matières

1 Contexte du Projet

Les échecs, en tant que jeu stratégique intemporel, attirent des joueurs de tous âges et niveaux. Avec l'avènement des technologies modernes, le développement d'un jeu d'échecs accessible en ligne, doté d'une intelligence artificielle (IA) compétitive et d'une interface conviviale, est une opportunité d'offrir une expérience enrichissante et moderne aux utilisateurs.

Le projet consiste à créer une plateforme de jeu d'échecs qui puisse fonctionner à la fois en mode local (sans connexion Internet) et en mode multijoueur (en ligne ou en réseau local). Le tout sera développé en Rust pour la robustesse du backend et avec un frontend basé sur des technologies web modernes.

2 Présentation de l'équipe de développement

2.1 Matthis Guillet

Étudiant en deuxième année à l'EPITA. Passionné d'informatique et de mathématiques depuis le collège, il aime découvrir de nouvelles branches de ces domaines et faire des projets pour appliquer la théorie. Il souhaite s'orienter dans l'IA et la robotique comme ingénieur ou chercheur.

2.2 Martin Madier

Étudiant en deuxième année à l'EPITA. Passionné par l'informatique et les nouvelles technologies, il aime découvrir de nouveaux domaines, et ce projet est une occasion de développer des compétences en IA et en programmation en Rust. De plus, c'est aussi une occasion de découvrir le fonctionnement d'une application web qui repose sur des requêtes API.

2.3 Martin Pasquier

Actuellement en deuxième année à l'EPITA, il est un étudiant passionné par l'informatique et la technologie. Il consacre une partie de son temps libre à des projets annexes, ce qui lui permet d'approfondir ses compétences techniques. Doté de bonnes connaissances en développement web, il a également un intérêt marqué pour gérer des projets avec Git. En tant que chef de groupe, il mettra également son temps à coordonner les différentes étapes du projet avec efficacité et rigueur.

2.4 Matteo Wermert

Étudiant en deuxième année à l'EPITA, une école d'ingénieurs en informatique, il a toujours été fasciné par les échecs. Ce jeu l'impressionne par sa richesse stratégique, sa profondeur et son élégance. Cela représente également pour lui une excellente opportunité d'apprendre un nouveau langage de programmation, le Rust. Ce projet lui permettra ainsi de concilier un sujet qui le passionne et une compétence technique essentielle pour son avenir.

3 Objectifs du projet

3.1 Objectifs Généraux

Pour créer une plateforme de jeux en ligne attrayante, nous allons développer un jeu d'échecs complet et fonctionnel, intégrer un panel varié de jeux disponibles, et fournir une intelligence artificielle capable de calculer des coups optimaux selon différents niveaux de difficulté. De plus, nous proposerons une interface web intuitive et performante pour garantir une expérience utilisateur agréable et fluide.

3.2 Objectifs Spécifiques

Pour offrir une expérience de jeu enrichissante, nous permettrons aux utilisateurs de jouer contre une intelligence artificielle ou un autre joueur, tout en offrant un mode multijoueur accessible via une connexion locale ou Internet. Nous proposerons également une interface web intuitive et performante, assurant une navigation fluide et rapide grâce à une architecture web moderne.

4 Architecture du Système

4.1 Backend (Rust)

Le *backend* de notre système est développé principalement en Rust, un langage de programmation réputé pour sa performance et sa sécurité. Nous utilisons le framework web *Actix-Web* pour structurer notre application. Les principales responsabilités du *backend* incluent la gestion des règles des échecs, assurant ainsi que chaque mouvement respecte les contraintes du jeu. De plus, il est chargé du calcul des mouvements optimaux de l'IA, permettant de proposer des défis adaptés aux différents niveaux de compétence des joueurs.

4.2 *Frontend* (Web)

Le *frontend* est développé en utilisant principalement *Rust*, *HTML*, *CSS* et *JavaScript*, avec le framework *Yew* pour structurer l'application. Ses responsabilités incluent l'affichage graphique de l'échiquier et des pièces, ainsi que la gestion des interactions utilisateur telles que le *drag-and-drop* et les clics.

4.3 Communication Client-Serveur

La communication entre le client et le serveur reposera sur une API dédiée, qui facilitera l'échange d'informations relatives à la création et à la gestion des parties. Cette API constituera le socle des interactions entre le serveur et les utilisateurs.

Afin d'assurer une expérience de jeu fluide et réactive, notamment dans le cadre des parties d'échec, nous intégrerons la technologie WebSocket. Celle-ci permettra d'établir une connexion bidirectionnelle entre le client et le serveur. Grâce à ce mécanisme, chaque joueur recevra instantanément les mises à jour relatives à l'état de la partie, lui permettant d'interagir sans latence et d'afficher dynamiquement les informations sur l'interface graphique. Lorsqu'un utilisateur rejoindra une partie, une connexion WebSocket sera établie avec le serveur.

Sur le plan technique, cette partie du projet sera mise en œuvre en Rust à l'aide des bibliothèques Tokio et Warp, qui offriront une infrastructure robuste et simple pour le serveur web. La gestion des connexions WebSocket sera assurée par la bibliothèque Tungstenite. L'association de ces technologies nous permettra de concevoir un système réactif et évolutif, capable de prendre en charge un grand nombre de connexions simultanées tout en garantissant une excellente réactivité du jeu.

5 Fonctionnalités

5.1 Fonctionnalités de Base

Notre système propose un environnement d'échecs capable d'interpréter un langage et d'interagir avec l'échiquier en conséquence. Il offre un affichage de l'échiquier en 2D avec plusieurs styles de plateaux d'échecs, permettant aux utilisateurs de personnaliser leur expérience visuelle. Les pièces se déplacent selon les règles officielles des échecs, et le système détecte automatiquement les situations d'échec, échec et mat, ainsi que les pats.

Les utilisateurs peuvent sauvegarder et reprendre leurs parties à tout moment. Le système propose un mode joueur contre joueur, disponible en local et en ligne, ainsi qu'un mode joueur contre IA avec différents niveaux de difficulté. Un timer est également intégré pour les parties chronométrées, ajoutant une dimension compétitive au jeu.

5.2 Fonctionnalités Avancées

Parmi les fonctionnalités avancées, le système inclut un historique des coups, permettant aux joueurs de revoir les actions passées. Le mode multijoueur est disponible en réseau local ou en ligne, offrant une flexibilité dans les options de jeu. Les utilisateurs peuvent revoir les parties et les coups joués, ce qui est utile pour l'analyse et l'amélioration des compétences.

Un mode de création de parties complet est proposé, avec des options telles que le niveau de l'IA et la durée de la partie. L'IA est capable de battre un joueur d'échecs de niveau supérieur à 2000, offrant un défi significatif même pour les joueurs expérimentés. De plus, l'IA peut conseiller le joueur lorsqu'il revoit sa partie, fournissant des indices précieux pour l'amélioration continue.

6 Contraintes Techniques

6.1 Backend

Le développement du backend est soumis à plusieurs contraintes techniques. Il doit être réalisé exclusivement en Rust, un langage choisi pour sa performance et sa sécurité. L'API développée en Rust doit respecter les normes HTTP, garantissant ainsi une compatibilité et une interopérabilité optimales avec les autres composants du système. De plus, le backend doit être conçu pour être performant, capable de gérer efficacement plusieurs connexions simultanées, assurant ainsi une expérience utilisateur fluide et sans interruption, même sous une charge élevée.

6.2 Frontend

Le frontend est conçu pour être compatible avec les principaux navigateurs web, notamment Chrome, Firefox, Edge et Safari, garantissant ainsi une accessibilité maximale pour les utilisateurs. L'interface est responsive, s'adaptant parfaitement à différents appareils, qu'il s'agisse d'ordinateurs de bureau, de tablettes ou de smartphones. Le temps de réponse pour les interactions est minimisé, offrant une expérience utilisateur fluide et réactive. De plus, le frontend est optimisé pour une faible consommation de ressources (RAM, GPU ou CPU), assurant une performance optimale sans surcharger les appareils des utilisateurs.

6.3 Hébergement

Notre application possède un seul point d'entrée, le serveur *Tokio*. Ce serveur est capable de gérer à la fois les requêtes HTTP et les connexions WebSocket. Il est donc possible de déployer notre application sur un ordinateur possédant rust et cargo.

Pour faciliter l'hébergement de notre application, nous avons choisi d'utiliser la technologie *Docker*, qui permet de créer des conteneurs logiciels légers et portables. Ces conteneurs sont des environnements isolés qui contiennent tout le nécessaire pour faire fonctionner une application, y compris les bibliothèques, les dépendances et les fichiers de configuration.

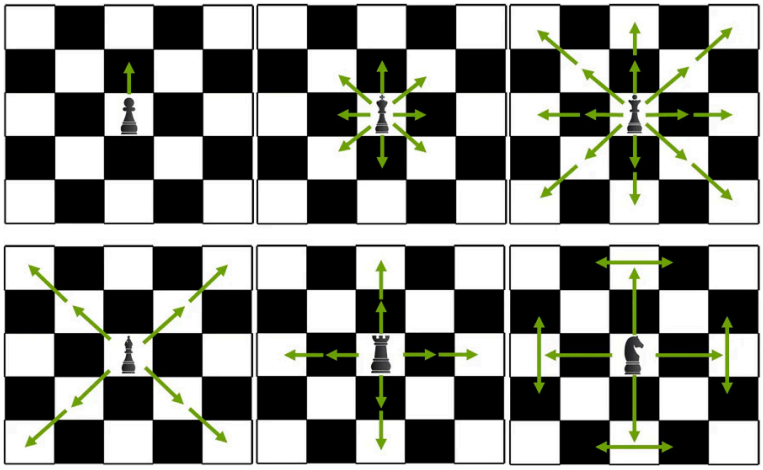
Avec seulement un fichier de configuration, il est possible de définir un conteneur qui peut être exécuté sur n'importe quel système d'exploitation qui prend en charge *Docker*. Cela permet de garantir que l'application fonctionnera de la même manière sur différents environnements.

7 Backend-Game

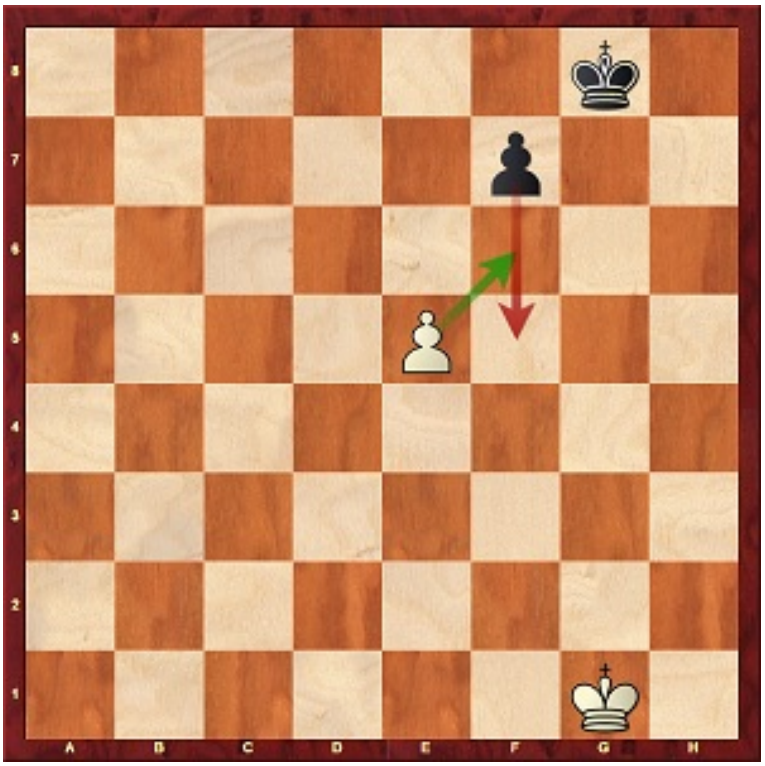
7.1 Fait

Nous avons implémenté plusieurs fonctionnalités essentielles pour le backend du jeu d'échecs. Les pions peuvent désormais être déplacés conformément aux règles du jeu, et les tours sont correctement gérées. Les mouvements basiques des pièces sont pris en charge, et la gestion du plateau assure une représentation précise de l'état de la partie.

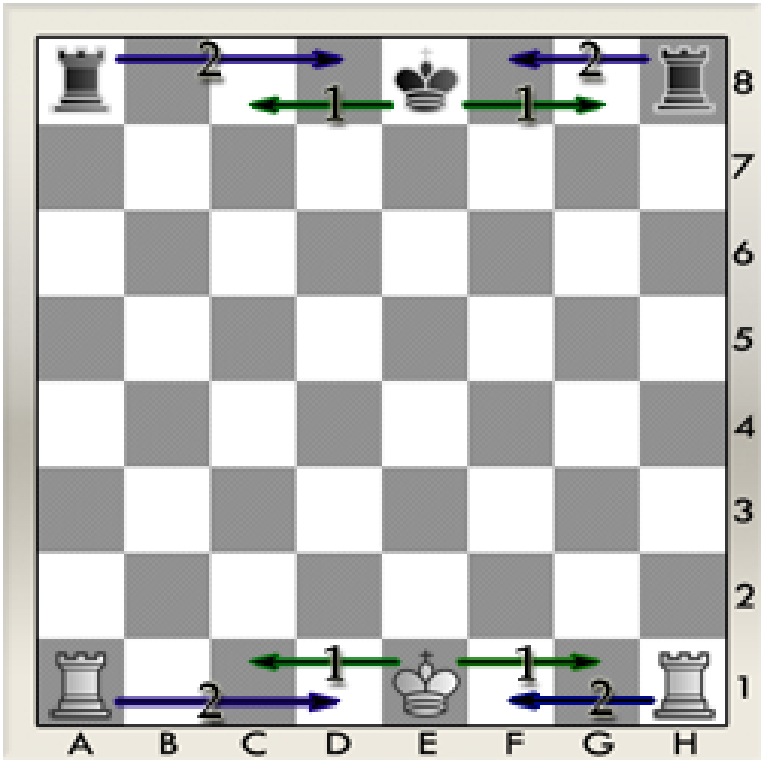
Les positions des pièces sont suivies avec précision, permettant une gestion fluide des déplacements. La gestion des mouvements est optimisée grâce à un parallélisme entre les positions sur le plateau et les pions stockés dans un vecteur à deux dimensions, une dimension pour chaque couleur. Cette structure permet de rapidement accéder aux pièces et de calculer les mouvements possibles de manière efficace.



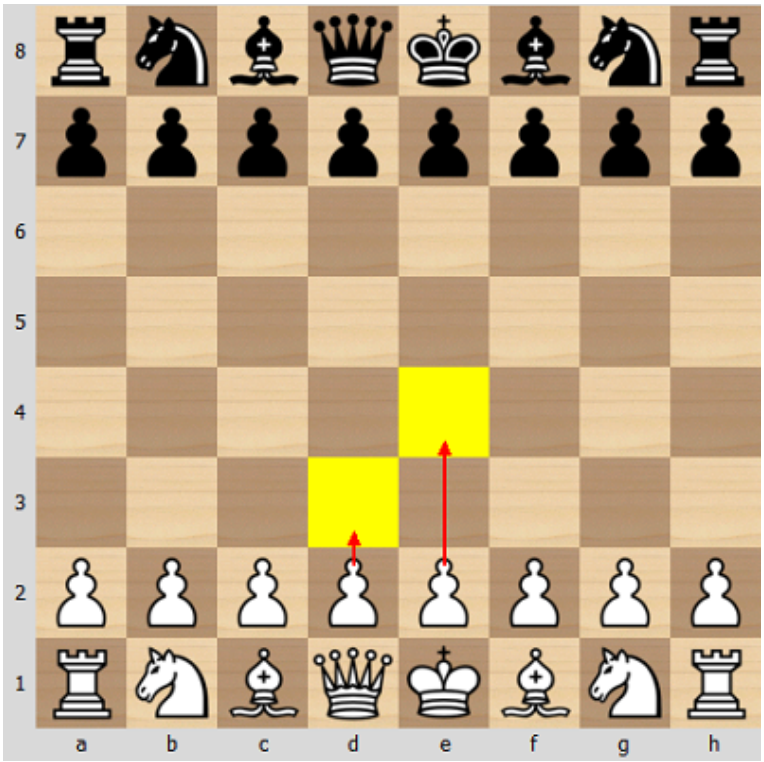
Pour le coup en passant, nous avons implémenté une logique qui détecte quand un pion adverse avance de deux cases à côté d'un pion grâce à notre historique, permettant ainsi de le capturer comme si le pion adverse n'avait avancé que d'une seule case. Cette règle est vérifiée après chaque mouvement de pion pour s'assurer qu'elle est appliquée correctement.



Le roque a été intégré en vérifiant plusieurs conditions : le roi et la tour impliqués n'ont pas bougé précédemment, il n'y a pas de pièces entre eux, et le roi n'est pas en échec, et ne passe pas par une case le lettant en echec. Si toutes ces conditions sont remplies, le roi et la tour sont déplacés simultanément pour compléter le roque.



Le mouvement double du pion a été mis en œuvre en permettant à un pion de se déplacer de deux cases vers l'avant lors de son premier mouvement, à condition que les deux cases soient libres. Cette règle est vérifiée lors de chaque tentative de déplacement d'un pion. Nous passons par la position du pion pour vérifier cela. En effet si le pion n'a pas bougé, il est à sa position initial



La mise en échec est détectée et gérée en vérifint si un pion adverse peut aller sur la

case du roi, offrant ainsi une expérience de jeu complète. Enfin, le système est capable de récupérer les mouvements possibles pour chaque pièce, à l'exception des situations d'échec et mat, qui nécessitent une logique supplémentaire.

7.2 Problématiques rencontrées

7.2.1 Gestion de la mémoire en *Rust*

Dans le développement du jeu d'échecs en Rust, on a rencontré plusieurs problématiques liées à la gestion de l'*ownership* et des emprunts.

Le principal défi a été de gérer l'utilisation simultanée de références mutables et immuables, ce qui a souvent provoqué des conflits d'emprunt à cause des règles strictes de Rust Borrow Checker.

On s'est aussi heurté l'immutabilité transitive, c'est-à-dire que dès qu'on empruntait une partie de l'échiquier en immuable, tout le reste devenait aussi immuable, ce qui compliquait les modifications croisées entre les pièces et les cases. On a donc dû trouver des solutions pour pouvoir accéder aux données de manière flexible tout en respectant les règles d'*ownership*. On a essayé plusieurs approches, comme l'utilisation de scopes pour limiter la durée de vie des emprunts ou le clonage temporaire des pièces pour libérer l'échiquier et pouvoir continuer à modifier le reste sans conflits.

On a aussi fait le choix de ne pas utiliser certains outils ne soit comme *RefCell* pour garder un code plus performant et idiomatique en Rust. Tout cela a révélé la complexité de la gestion des références dans un projet où les données sont très interdépendantes, tout en cherchant à maintenir un code lisible et performant.

7.2.2 Complexité d'implémentation des règles échecs

Lors de notre implémentation, l'optimisation a été notre principal défi, en lien direct avec les problématiques abordées précédemment. Nous avons d'abord utilisé le clonage de structure pour sa simplicité, mais sa complexité en O(n)O(n) a rapidement révélé des limitations de performance.

Pour y remédier, nous avons opté pour les emprunts et le système d'*ownership* de Rust, ce qui a amélioré la gestion de la mémoire. Cependant, les restrictions strictes de Rust ont rendu cette approche difficile à appliquer, notamment pour les mouvements complexes du jeu d'échecs.

Finalement, nous avons adopté une solution hybride en combinant clonage et emprunts. Le clonage a été utilisé uniquement lorsque son impact sur les performances était négligeable, tandis que les emprunts ont été réservés aux parties critiques du code. Cette combinaison nous a permis d'optimiser efficacement les performances tout en maintenant une gestion sécurisée de la mémoire.

7.2.3 Optimisation

Lors de notre implémentation, l'optimisation a été notre principal défi, en lien direct avec les problématiques abordées précédemment. Nous avons d'abord utilisé le clonage de structure pour sa simplicité, mais sa complexité en O(n)O(n) a rapidement révélé des limitations de performance.

Pour y remédier, nous avons opté pour les emprunts et le système d'*ownership* de Rust, ce qui a amélioré la gestion de la mémoire. Cependant, les restrictions strictes de Rust ont rendu cette approche difficile à appliquer, notamment pour les mouvements complexes du jeu d'échecs.

Finalement, nous avons adopté une solution hybride en combinant clonage et emprunts. Le clonage a été utilisé uniquement lorsque son impact sur les performances était négligeable, tandis que les emprunts ont été réservés aux parties critiques du code. Cette combinaison nous a permis d'optimiser efficacement les performances tout en maintenant une gestion sécurisée de la mémoire.

7.3 Retards Constatés

Nous n'avons rencontré aucun retard au cours du développement. Malgré cela, lors de l'optimisation du plateau, qui était initialement prévue pour la deuxième soutenance, nous avons dû faire des compromis sur certaines règles. En effet, certaines d'entre elles ne fonctionnaient pas correctement dans les délais impartis, ce qui nous a conduits à les mettre temporairement de côté.

Ce choix stratégique nous a permis de respecter le calendrier sans compromettre la qualité générale du projet. Toutefois, il est important de noter que ces règles incomplètes représentent une opportunité d'amélioration pour les futures versions du jeu. En réintégrant ces éléments manquants et en optimisant leur fonctionnement, nous pourrions atteindre une complétude parfaite des mécaniques de jeu tout en conservant les performances actuelles.

Ainsi, bien que des ajustements aient été nécessaires, l'ensemble des objectifs principaux a été atteint dans les délais, démontrant notre capacité à gérer efficacement les priorités tout en maintenant une progression constante du projet.

7.4 Améliorations Possibles

Le jeu d'échecs n'est pas encore optimisé au maximum. En effet, certaines opérations sont effectuées plusieurs fois sans être mises en cache, ce qui entraîne des calculs redondants. Par exemple, l'ensemble des mouvements possibles pour une pièce spécifique qui n'a pas bougé est recalculé à chaque nouveau tour, alors qu'il pourrait être stocké pour être réutilisé ultérieurement.

Pour remédier à cela, nous avons commencé à implémenter un nouveau système d'optimisation. Toutefois, par manque de temps, nous avons dû faire des compromis en négligeant certains coups particuliers qui ne fonctionnaient pas correctement avec ce système.

Cela signifie que certains scénarios ne sont pas encore entièrement pris en charge, limitant ainsi la complétude du jeu.

L'une des améliorations majeures à envisager serait donc d'intégrer ces coups manquants afin de garantir une expérience de jeu totalement fluide et conforme aux règles classiques des échecs. Cela permettrait non seulement d'optimiser les performances en réduisant les calculs inutiles, mais aussi de s'assurer que toutes les situations de jeu soient correctement gérées.

De plus, d'autres pistes d'optimisation pourraient être explorées, comme l'implémentation de mécanismes de mise en cache plus sophistiqués ou l'amélioration de l'algorithme de recherche des coups. En procédant ainsi, le jeu gagnerait en efficacité tout en offrant une expérience plus fluide et réactive pour les joueurs.

8 Frontend-Web

8.1 État actuel

Aujourd'hui, le site propose une page d'accueil sobre avec trois boutons permettant de naviguer facilement entre les différents onglets du site :

- **L'onglet jeu** : il contiendra les différentes fonctionnalités du projet. Pour l'instant, nous avons un échiquier simple avec les pièces correctement disposées. Il est possible de déplacer les pièces, mais la communication avec l'API n'est pas encore totalement implémentée. Ainsi, les coups ne sont pas vérifiés, et l'utilisateur peut déplacer n'importe quelle pièce sans restriction.
- **L'onglet présentation** : cet espace offre une description du projet, une présentation des membres et de leurs rôles respectifs.
- **L'onglet téléchargement** : ici, les utilisateurs peuvent télécharger le projet ainsi que le rapport de soutenance.

8.2 Problématiques rencontrées

Pour le développement du site web, nous utilisons *Yew*, un framework permettant de générer du HTML à partir de Rust. Ce choix a été fait afin d'utiliser Rust au maximum dans notre projet. Cependant, cette approche nous a considérablement ralentis et a empêché la finalisation des liens avec l'API. Ce manque d'intégration constitue actuellement le principal problème du site.

Nous avons déjà plusieurs pistes pour résoudre cette problématique, mais cela nécessitera du temps et une refonte partielle de l'architecture du site.

8.3 Améliorations possibles

Une fois le site opérationnel et la communication avec l'API fonctionnelle, nous pourrons ajouter d'autres fonctionnalités, telles que :

- Le changement de plateau,
- La création de parties,

- Et d'autres améliorations visant à enrichir l'expérience utilisateur.

9 API

9.1 État d'avancement

L'API n'est pas encore fonctionnelle. Nous avons commencé à réfléchir à son implementation et à sa structure mais nous avons fait le choix d'attendre que le projet avance avant de l'implémenter. En effet, nous avons préféré nous concentrer sur le système de jeu et sur l'interface dans un premier temps.

L'API sera implémentée en *Rust* avec la bibliothèques *Tokio*, celle-ci permettant de gérer de manière asynchrone les appels réseaux. Nous l'avons choisi pour sa simplicité d'utilisation et pour sa grande flexibilité. Plus tard, nous integrerons également la tecnologie *WebSockets* pour permettre une communication en temps réel entre le serveur et les clients.

9.2 Problématiques rencontrées

La création de l'API n'a pas encore commencé, nous n'avons donc pas rencontré de problématiques particulières lors du développement de celle-ci. Cependant, il n'a pas été simple de trouver une méthode fonctionel pour distribuer le *front-end* et assurer la communication entre le serveur et les clients avec la même technologie.

9.3 Retards Constatés

Nous avons pris du retard sur ce point car nous avons préféré nous concentrer sur le système de jeu et sur l'interface.

10 Organisation du projet

10.1 répartition des tâches

- Matteo : *Frontend-Web*
- Matthis : *Backend-Game/IA*
- Martin M. : *Backend-Game/IA*
- Martin P. : *API*

10.2 Tableau d'avancement

Tâches	1ère soutenance	Prévision 2ème soutenance
Jeu	70%	100%
Règles	70%	100%
IA	25%	60%
Frontend	25%	60%
Site-Web	20%	55%
IA :	20%	60%
API : 25%	50%	100%

TABLE 1 – Planning d'avancement du projet