Rapport : Collectible Card Game

Hejun CAO, Haotian XUE

Octobre 2024

Contents

1 Introduction

Avec l'essor de la blockchain, les **jeux de cartes à collectionner (CCG)**, comme Pokémon et Magic: The Gathering, ont trouvé une nouvelle forme de gestion et d'échange. Ce projet a pour objectif de créer une plateforme décentralisée sur la blockchain Ethereum, permettant aux utilisateurs de créer, collecter et échanger des cartes sous forme de **NFT**.

L'infrastructure proposée inclut des contrats intelligents pour gérer les cartes, un backend pour interagir avec la blockchain et un frontend pour offrir une interface simple aux utilisateurs.

2 Analyse des besoins

2.1 Besoins fonctionnels

- Création et gestion des cartes en tant que NFT.
- Système d'achat et d'échange sécurisé sur la blockchain.
- Interface permettant la connexion des utilisateurs via Metamask pour visualiser et échanger des cartes.

2.2 Besoins techniques

- Utilisation de la norme **ERC-721** pour les NFT, garantissant l'unicité des cartes.
- Intégration fluide avec **Metamask** pour la gestion des transactions utilisateurs.
- Backend capable de communiquer avec les contrats intelligents et de récupérer les données de la blockchain.

3 Technologie utilisée

3.1 Environnement de développement

- **HardHat** : Simulation de blockchain locale pour le développement et les tests.
- **Node.js** et **Yarn** : Utilisés pour la gestion des dépendances et le développement backend.

- **Metamask** : Extension permettant aux utilisateurs de gérer leurs transactions de manière sécurisée.

3.2 Contrats intelligents et ERC-721

Le projet repose sur l'implémentation d'**ERC-721** pour gérer les cartes en tant que NFT. Chaque carte est identifiée par un **tokenId** unique et contient des métadonnées pour stocker ses informations (nom, image).

4 Implémentation des contrats intelligents

4.1 Conception des contrats

Les contrats intelligents constituent le cœur de ce projet. L'objectif principal est de gérer la création de cartes sous forme de jetons non fongibles (NFT) basés sur la norme ERC-721, permettant ainsi de garantir l'unicité et la transférabilité de chaque carte. La conception des contrats est modulaire, avec plusieurs composants clés :

- **Booster.sol** : Ce contrat gère l'achat et l'ouverture de boosters, qui sont des paquets de cartes. Chaque booster est un NFT distinct et contient un ensemble de cartes aléatoires. Le contrat émet des événements pour chaque achat et ouverture de booster.

- **Collection.sol** : Ce contrat est responsable de la gestion des collections de cartes. Il permet la création et la gestion de cartes sous forme de NFT. Il intègre également un mécanisme permettant de frapper (mint) des cartes aléatoires lorsqu'un booster est ouvert.

- **Main.sol** : Ce contrat centralise la gestion des différentes collections. Bien que dans sa forme actuelle, il ne comporte que des fonctionnalités de base, il pourrait être développé pour gérer plusieurs collections simultanément et offrir un point d'entrée unique pour les utilisateurs.

- **XueCaoToken.sol** : Ce contrat implémente un jeton ERC-20 appelé XueCao. Il pourrait être utilisé comme devise pour acheter des cartes ou des boosters dans l'écosystème du jeu.

- **Lock.sol** : Bien que ce contrat ne soit pas directement lié à la gestion des cartes, il implémente un mécanisme de verrouillage de fonds qui peut être utilisé pour des fonctionnalités futures comme des coffres ou des récompenses à débloquer.

Ces contrats sont conçus pour interagir entre eux de manière transparente, permettant aux utilisateurs d'acheter des boosters, d'ouvrir ces boosters et de recevoir des cartes directement sous forme de NFT.

4.2 Fonctions principales du contrat

Le contrat principal contient trois fonctions critiques pour la gestion des cartes :

- **mint** : Cette fonction permet à l'administrateur du contrat de créer de nouvelles cartes sous forme de NFT et de les attribuer à des utilisateurs. Le processus de minting garantit que chaque carte possède un identifiant unique (‘tokenId‘) et est associée à des métadonnées (comme une image ou un nom).

- **assign** : Cette fonction permet à l'administrateur d'attribuer un ensemble de cartes à un utilisateur spécifique, souvent en réponse à l'achat ou à l'ouverture d'un booster.

- **openBooster** : Cette fonction, implémentée dans le contrat **Booster.sol**, permet aux utilisateurs d'ouvrir un booster qu'ils possèdent. Cela déclenche l'appel à la fonction **mintRandomCards** dans le contrat **Collection.sol**, qui génère un ensemble de cartes aléatoires pour l'utilisateur.

4.3 Le processus de minting

Le processus de minting est essentiel pour garantir l'unicité de chaque carte dans le jeu. Lorsqu'un utilisateur ouvre un booster, la fonction **openBooster** est déclenchée, ce qui brûle le booster et appelle la fonction **mintRandomCards** dans le contrat **Collection.sol**. Cette fonction frappe un certain nombre de cartes aléatoires (défini dans le booster) pour l'utilisateur et associe à chaque carte un ‘tokenId‘ unique ainsi qu'une URL d'image (ou d'autres métadonnées).

- **Unicité** : Chaque carte est associée à un ‘tokenId‘ unique, garantissant qu'aucune carte n'est identique à une autre. Le ‘tokenId‘ est incrémenté à chaque frappe, ce qui assure que chaque carte reçoit un identifiant distinct.

- **Gestion des métadonnées** : Les métadonnées associées à chaque carte incluent des informations importantes telles que l'image de la

carte. Bien que pour des raisons de simplicité, nous utilisons actuellement une URL fixe pour les métadonnées, il est prévu de mettre en place un système de sélection aléatoire pour offrir une plus grande diversité dans les cartes générées.

Le processus de minting garantit que les utilisateurs reçoivent des cartes uniques à chaque interaction avec le contrat, renforçant l'aspect de collection et de rareté propre aux jeux de cartes à collectionner.

5 Intégration entre le frontend et le backend

5.1 Conception de l'API

Le backend joue un rôle essentiel en tant qu'intermédiaire entre les utilisateurs et la blockchain, facilitant l'accès aux données des cartes à collectionner et aux événements de la blockchain. L'API expose des points de terminaison permettant de récupérer des informations sur les cartes possédées par les utilisateurs, ainsi que les boosters achetés. Voici les principales caractéristiques de l'API :

- **Récupération des cartes des utilisateurs** : Un point de terminaison API (‘/api/user-cards‘) permet de récupérer les cartes achetées par un utilisateur spécifique. Cela repose sur l'analyse des événements **CardPurchased** émis par le contrat **Collection.sol**. L'API écoute ces événements et stocke les enregistrements pertinents dans une mémoire temporaire.
- **Suivi des événements** : Le backend utilise la bibliothèque **ethers.js** pour se connecter à un fournisseur Ethereum local (via HardHat) et surveille les événements de type **CardPurchased** et **BoosterPurchased**. Lorsqu'un événement est détecté, l'API extrait les données nécessaires, telles que l'adresse de l'acheteur, l'ID du jeton et l'URI du jeton, et les enregistre pour un accès futur.

L'API expose également des points de terminaison pour obtenir des données de carte à partir de sources externes, comme l'API Pokémon TCG, et les associer aux jetons non fongibles sur la blockchain.

5.2 Implémentation du frontend

Le frontend est une interface utilisateur qui permet aux utilisateurs de se connecter à leur portefeuille Ethereum (via Metamask), d'interagir avec les

contrats intelligents et de visualiser leurs cartes et boosters. Voici les principales fonctionnalités mises en œuvre :

- **Connexion au portefeuille** : Le frontend utilise **ethers.js** pour permettre aux utilisateurs de se connecter à leur portefeuille Metamask. Une fois connecté, l'utilisateur peut voir ses cartes et ses boosters, et effectuer des transactions en utilisant son portefeuille.

- **Achat de cartes** : Dans le composant 'Shop.tsx', une fonction 'buyCard' permet aux utilisateurs d'acheter des cartes en envoyant une transaction à l'adresse du contrat **Collection.sol**. Le frontend recueille les informations de l'utilisateur et le prix de la carte, puis déclenche la fonction de frappe du contrat intelligent.

- **Achat de boosters** : Le frontend propose également une fonctionnalité d'achat de boosters, mise en œuvre dans 'Shop.tsx'. Lorsqu'un utilisateur achète un booster, le frontend envoie une transaction au contrat **Booster.sol** pour créer un booster unique sous forme de NFT.

- **Affichage et gestion des cartes** : Les cartes possédées par l'utilisateur sont récupérées via l'API ('/api/user-cards') et affichées dans une galerie. L'utilisateur peut voir les images des cartes et leurs informations de métadonnées.

- **Affichage et gestion des boosters** : Les utilisateurs peuvent voir les boosters qu'ils possèdent et les ouvrir en déclenchant une transaction vers le contrat **Booster.sol**. Lors de l'ouverture d'un booster, des cartes aléatoires sont générées et attribuées à l'utilisateur.

Technologies utilisées pour le frontend

Pour implémenter l'interface utilisateur, nous avons utilisé les technologies suivantes :

- **React** : Un framework JavaScript pour créer des interfaces utilisateur interactives. React est utilisé pour construire les composants de l'application, gérer les états et les événements, et rendre dynamiquement les cartes et les boosters.

- **ethers.js** : Une bibliothèque légère pour interagir avec Ethereum. Nous l'avons utilisée pour connecter Metamask, récupérer les informations des contrats intelligents et envoyer des transactions.

- **CSS modules** : Pour styliser les composants et créer une interface utilisateur cohérente, nous avons utilisé des modules CSS, offrant un style localisé à chaque composant.
- **Express.js** : Pour le backend, nous avons utilisé Express.js pour exposer des points de terminaison RESTful et gérer les requêtes API depuis le frontend.

Grâce à cette intégration entre le frontend et le backend, les utilisateurs peuvent interagir facilement avec la blockchain, acheter des cartes et des boosters, et gérer leur collection de cartes directement depuis leur navigateur.

6 Extensions de fonctionnalités

6.1 Intégration de Pokémon TCG

L'intégration de l'API Pokémon TCG permet d'ajouter des données réelles aux cartes NFT. Chaque carte NFT reçoit des métadonnées provenant de cette API, incluant le nom et l'image des cartes, renforçant l'aspect collection.

6.2 Système de boosters

Les boosters sont implémentés comme des NFT contenant plusieurs cartes. Lorsqu'un utilisateur ouvre un booster, le NFT est brûlé et des cartes aléatoires sont générées via le contrat **Collection.sol**.

7 Défis et solutions

7.1 Sécurité des contrats intelligents

Nous avons utilisé **OpenZeppelin** pour sécuriser les contrats, en particulier le module ‘Ownable‘, qui limite l'accès aux fonctions sensibles aux seuls propriétaires.

7.2 Intégration frontend-backend

La gestion des événements en temps réel via **ethers.js** a permis une communication fluide entre le frontend et la blockchain. Les données sont synchronisées pour que les utilisateurs puissent visualiser leurs cartes et boosters instantanément.

7.3 Expérience utilisateur

Pour garantir une bonne expérience utilisateur, nous avons optimisé la gestion des erreurs et l'interface avec **Metamask**, facilitant ainsi la connexion et les transactions.

8 Conclusion

Le projet a permis de créer une plateforme fonctionnelle pour la gestion de cartes à collectionner en utilisant des NFT. L'intégration de Pokémon TCG et la gestion des boosters ont été implémentées avec succès.

Dans le futur, des fonctionnalités telles qu'un marché décentralisé ou des tournois pourraient être envisagées pour enrichir l'expérience utilisateur.