.

0.1 Introduction

Les jeux de cartes à collectionner (JCC) ont toujours été un passe-temps populaire pour les amateurs de jeux, et la montée en puissance de la technologie blockchain a apporté des innovations passionnantes à ce domaine. Ce projet a pour objectif de créer un jeu de cartes à collectionner numérique basé sur la blockchain Ethereum, en utilisant des jetons non fongibles (NFT) pour représenter les cartes du jeu.

0.1.1 Contexte et Pertinence

Dans un monde numérique en constante évolution, les JCC numériques offrent de nouvelles possibilités passionnantes pour les joueurs. La blockchain Ethereum, avec sa technologie NFT, permet de garantir la rareté et la propriété des cartes du jeu, offrant ainsi une expérience unique aux collectionneurs et aux joueurs. De plus, les NFT permettent aux joueurs de posséder réellement leurs cartes et de les échanger en toute sécurité avec d'autres joueurs.

0.1.2 Objectifs du projet

Les principaux objectifs de cette étude sont les suivants :

— Concevoir et développer un contrat intelligent (smart contract) NFT sur la blockchain Ethereum pour représenter les cartes du jeu. (Objectif atteint)
— Créer une interface utilisateur conviviale pour permettre aux joueurs de visualiser, échanger et posséder des cartes NFT. (Objectif atteint)
— Intégrer des cartes Pokémon de l'API Pokémon Trading Card Game (TCG) dans le jeu, permettant ainsi aux joueurs de collectionner des cartes du monde réel sous forme de NFT. (Objectif atteint)
— Proposer une solution pour créer des boosters (paquets de cartes) contenant des cartes NFT aléatoires. (Objectif non atteint)

0.2 Conception et réalisation

Dans cette section, nous détaillerons la méthodologie utilisée pour concevoir et développer notre jeu de cartes à collectionner (JCC) basé sur la blockchain et les jetons non fongibles (NFT). Nous expliquerons les étapes clés impliquées dans la réalisation de ce projet ambitieux.

0.2.1 Conception des Contrats Intelligents Main et Collection

La première étape de notre méthodologie consiste à concevoir le contrat intelligent Collection, puis le contrat Main qui représentera les cartes de notre JCC. Pour ce faire, nous utiliserons le langage de programmation Solidity, spécialement conçu pour la blockchain Ethereum. La conception du contrat

— Fonctionnalité de transfert de cartes entre les joueurs (non implémenter à cause de probleme de gas) après une battle.
— Possibilité d'ouvrir des boosters pour obtenir de nouvelles cartes NFT.

L'interface utilisateur sera conçue de manière conviviale, avec une attention particulière à l'esthétique et à la facilité d'utilisation.

0.2.3 Intégration de l'API Pokémon TCG

Pour offrir une expérience encore plus passionnante aux joueurs, nous intégrerons des cartes Pokémon réelles à notre JCC. Pour ce faire, nous utiliserons l'API Pokémon Trading Card Game (TCG) pour accéder à un vaste ensemble de cartes Pokémon. Les étapes d'intégration de l'API comprendront :

— Mise en place des appels à l'API pour récupérer des données sur les cartes Pokémon.
— Conversion des données en cartes NFT conformes à notre contrat intelligent.
— Attribution des cartes Pokémon aux joueurs.

0.2.4 Gestion de Projet et Test

Pour garantir le succès de notre projet, nous mettrons en place une gestion de projet rigoureuse. Nous utiliserons des outils de suivi pour surveiller notre avancement et nous assurer que les délais sont respectés. De plus, nous effectuerons des tests intensifs pour vérifier la sécurité et la fonctionnalité de notre contrat intelligent NFT et de l'interface utilisateur.

0.2.5 Gestion et mise en place de la blockchain :

Pour commencer, nous avons préféré utiliser un réseau de test au lieu d'un réseau local, afin de nous rapprocher au plus près de l'expérience professionnelle et d'acquérir une pratique similaire à celle d'une situation de travail réelle. Nous avons opté pour le réseau Sepolia pour suivre les transactions et les déploiements, en utilisant EtheriumScan. De plus, nous avons fait usage d'Alchemy, un service BaaS, qui permet d'accéder à la blockchain Ethereum sans avoir à gérer notre propre nœud Ethereum. Alchemy propose des API Ethereum et des outils qui simplifient la création, le déploiement et la gestion d'applications blockchain de manière plus efficace, offrant ainsi de nombreux avantages. Et nous avons fait appel au site https ://sepoliafaucet.com/ pour alimenter nos comptes d'ETH.

0.2.6 Contrats et fonctions :

Dans un premier temps, nous avions envisagé la création de plusieurs contrats, y compris un contrat principal, une collection et des NFTs. Cependant, en raison de multiples tentatives infructueuses et de contraintes de temps, nous n'avons pas pu mener une enquête en profondeur ni saisir pleinement le processus requis. Par conséquent, nous avons opté pour une approche plus souple, prenant quelques libertés par rapport aux étapes détaillées explicitement dans la description du projet. Pour une application DApp simple, il est généralement recommandé de déployer un seul contrat intelligent si les fonctionnalités de l'application peuvent être gérées efficacement de cette manière. Cela simplifie la structure de l'application, réduit les frais de déploiement et facilite la gestion globale de l'application. Les fonctions principales implémentés :

1- constructor(string memory $_name, stringmemory_symbol$) :
$leconstructeurducontratERC-721estutilispourinitialiserlenometlesymboleducontrat.$
$2.'mint(stringmemorytokenURI, addressuser)'$: $Cettefonctionpermetdecreretdedlivrerunnou$

3. 'getAllNfts() external view returns(string[] memory)' : Cette fonction renvoie la liste des URIs de tous les NFTs créés jusqu'à présent.

4. 'setAttributes(string memory name, uint256 cardCount) external' : Cette fonction permet de définir le nom de la collection et le nombre total de cartes de la collection.

5. 'getCardMetadata(uint256 tokenId) external view returns (string memory)' : Cette fonction renvoie les métadonnées associées à un NFT donné, identifié par son 'tokenId'.

6. 'getCardsMetadataLength() external view returns (uint256)' : Cette fonction renvoie le nombre total de NFTs créés jusqu'à présent, c'est-à-dire la longueur de la liste '$_tokenList'$.

7. 'transferCard(address to, uint256 tokenId) external' : Cette fonction permet de transférer un NFT de l'expéditeur ('msg.sender') vers une autre adresse ('to'). Elle effectue des vérifications pour s'assurer que le transfert est autorisé.

8. 'isApprovedOrOwner(address spender, uint256 tokenId) internal view returns (bool)' : Cette fonction vérifie si l'expéditeur ('spender') est autorisé à effectuer des actions sur un NFT donné, identifié par son 'tokenId'. Elle vérifie si l'expéditeur est le propriétaire du NFT ou s'il a été approuvé pour effectuer des actions sur ce NFT.

9. 'event CardAdded(uint256 tokenId, string metadata)' : Cet événement est déclenché lorsqu'un nouveau NFT est créé, fournissant le 'tokenId' du NFT créé et les métadonnées associées.

Le contrat semble bien adapté pour la création, la gestion et le transfert

de NFTs, avec la possibilité de stocker des métadonnées associées à chaque NFT et de renvoyer des informations sur les NFTs créés."

0.2.7 Gestion des données :

Afin de minimiser les coûts des transactions, nous avons choisi de stocker certaines données localement sous forme de fichiers JSON. Nous avons identifié six collections distinctes à utiliser dans ce projet, et donc nous récupérons, au lancement du projet, les ensembles de données disponibles depuis l'API, en ne collectant que les informations essentielles. Nous aurions pu récupérer toutes les collections, cependant, étant donné que ce projet est destiné à un contexte éducatif, nous mettons l'accent est mis sur la méthodologie plutôt que sur la quantité brute d'informations. Nous avons également créé un fichier JSON dans lequel nous stockons certaines données relatives à nos utilisateurs. Cela est essentiel pour vérifier à chaque connexion si l'utilisateur se connecte pour la première fois ou s'il s'agit d'un ancien utilisateur. En fonction de cette information, nous pouvons lui attribuer des cartes (la logique d'attribution des cartes sera expliquée dans la partie suivante).

0.2.8 Partie Off-Chain :

Attribution des cartes au départ

: Lorsque l'utilisateur se connecte sur notre site, nous vérifions s'il s'agit de sa première connexion. Si c'est le cas, nous attribuons une carte à l'utilisateur en utilisant deux fonctions aléatoires. La première fonction se base sur les noms des collections disponibles dans la liste des collections, tandis que la deuxième fonction s'appuie sur la plage allant du nombre minimum (différent de zéro) au nombre total de cartes disponibles dans la collection sélectionnée. Ces deux informations nous permettent de déterminer la carte que nous allons attribuer à notre utilisateur.

Récupération de toutes les cartes par collection

: L'utilisateur a la possibilité de parcourir les cartes appartenant à n'importe quelle collection disponible sur l'application et détenue par d'autres utilisateurs. Il peut les consulter et même lancer des combats contre ces cartes. Dans nous avons effectué des tri de cartes selon les collections.

Récupération des cartes d'un utilisateur en particulier connecté sur Metamask :

En nous basant sur les données d'un utilisateur récupérées lors de sa connexion à Metamask, nous extrayons ses cartes.

Le Booster :

Si l'utilisateur parvient à gagner 5 cartes, il reçoit un booster de 3 cartes attribuées de manière aléatoire, en utilisant le même algorithme que celui employé lors de l'attribution initiale des cartes lors de la première connexion sur le site. Il est important de noter que le paramètre "booster" est l'un des paramètres que nous stockons localement lors de la création du compte. Initialement, il est défini à zéro, puis il passe à 3. Au fur et à mesure de l'utilisation du booster, sa valeur se décrémente. Il revient à l'utilisateur de décider quand l'utiliser selon ses préférences.

Lancer des batles :

Nous avons impmementer une interface pour une bataille entre un joueur et un ennemi. Voici une explication brève du fonctionnement :

Deux Pokémon sont affichés, un représentant le joueur (votre Pokémon) et l'autre représentant l'ennemi . Les barres de santé de chaque Pokémon sont également affichées.https ://www.overleaf.com/project/65451110c3d1b5b64e3633a3

Le joueur a des options pour choisir une action :

Attaquer : Le joueur peut choisir l'une des trois attaques disponibles. Chaque attaque inflige un montant de dégâts aléatoires à l'ennemi. Guérir : Le joueur peut choisir de guérir son Pokémon, ce qui augmentera la santé de son Pokémon. L'ennemi choisit automatiquement une action en fonction de sa santé :https ://www.overleaf.com/project/65451110c3d1b5b64e3633a3

Si la santé de l'ennemi est élevée, il a une chance d'attaquer le joueur. Si la santé de l'ennemi est moyenne, il peut choisir d'attaquer ou de se soigner. Si la santé de l'ennemi est basse, il a plus de chances de se soigner. Les actions du joueur et de l'ennemi sont déterminées par des fonctions qui calculent les dégâts et les guérisons.

Les résultats de chaque action sont affichés dans un journal de combat sous forme de logs. Le journal affiche qui a attaqué qui et combien de dégâts ont été infligés.

La bataille continue jusqu'à ce que l'un des Pokémon atteigne une santé nulle ou que le joueur choisisse de se soigner après avoir perdu.

Lorsque la bataille est terminée, un message s'affiche pour indiquer si le joueur a gagné ou perdu.

0.3 Conclusion

Le projet a été une expérience intéressante, mais il y a eu des défis à relever en raison de l'absence de clarté dans la définition du sujet. Bien que le projet visait à créer une application liée à la collection de cartes Pokémon, il manquait des détails spécifiques sur les fonctionnalités requises et les objectifs à atteindre. On a passé la majeur partie de notre temps à essayer de comprendre le sujet et les technologies impliqués. En fin de compte, le projet nous a permis de développer ses compétences sur les technologies blockchain et NFT. Nous avons acquis une expérience précieuse dans la résolution de problèmes, la collaboration et la prise de décisions techniques.