

SORBONNE UNIVERSITÉ

DÉVELOPPEMENT DES ALGORITHMES D'APPLICATION
RÉTICULAIRE

5 novembre 2023

Blochain and Smart contract project

Collectible Card Game

Auteurs :

Daniel SIMA
Walter ABELES

Spécialité :

Science et Technologie
du Logiciel
(STL)



Table des matières

1 Partie introductive	2
1.1 Partie onchain	2
1.2 Partie offchain	2
2 Partie technique	3
2.1 Architecture du système	3
2.2 Account	3
2.3 Store	4
2.4 Boosters	6
3 Tests de performance	7
4 Conclusion	8
Références	8

1 Partie introductive

Dans le cadre de l'UE DAAR, ce projet consiste en l'implémentation d'un Jeu de Cartes à Collectionner (CCG) de manière décentralisée, sur Ethereum avec **Solidity**. Les cartes sous forme de NFT, peuvent être achetées et échanger avec les autres joueurs. Le joueur peut aussi ouvrir des boosters pour gagner de nouvelles cartes. La partie **onchain** sera programmée sur **Solidity** et la partie **offchain** en **JavaScript** en utilisant le framework **React**.

1.1 Partie onchain

Tout d'abord, il nous faut implémenter un contrat capable de créer des collections, chaque collection, ou ensemble, est composé d'un nom et d'un nombre de cartes. Un NFT est un token mettant en œuvre la norme ERC-721. Chaque carte sera un NFT issu d'une collection spécifique. Dans un premier temps, nous implémentons le numéro de la carte, un identifiant unique et un champ img dans les métadonnées du NFT. Le contrat principal a un propriétaire qui pourra créer des nouvelles cartes et attribuer des cartes à un utilisateur sélectionné dans une collection spécifique.

De plus, nous intégrons l'API Pokémons TCG [3] qui nous permet de récupérer des ensembles de cartes Pokémons. Pour ce qui est de la création des boosters, nous le gérons via le Smart Contract, en utilisant de l'aléatoire avec un bloc de temps et avec les adresses. L'enjeu dans cette partie est de créer une fonction dans notre contrat principal qui permet aux utilisateurs de récupérer les boosters NFT. Cette fonction doit garantir l'intégrité des cartes contenues dans le booster, ce qui signifie que les cartes doivent correspondre exactement à celles générées lors de la création du booster hors ligne. C'est pour cela que nous mettons en place des mécanismes pour garantir la sécurité et l'unicité des boosters NFT afin d'éviter la fraude et la duplication.

Pour une meilleure compréhension de la programmation de contrats, nous avons utilisé la plateforme Crypto Zombies [1] et pour se lancer dans programmation en **Solidity** nous avons tout d'abord commencé sur Remix - Ethereum IDE [2].

1.2 Partie offchain

Nous développons sous **React** une interface utilisateur employant le format JSON de l'API Pokémons pour visualiser toutes les cartes possédées par l'utilisateur et permettant de les manipuler. Cette interface doit contenir les fonctionnalités suivantes :

- Les utilisateurs auront quelques NFT après que le propriétaire en ait créé pour eux.
- Les utilisateurs pourront voir leurs NFT en se connectant à l'interface utilisateur
- Les utilisateurs récupéreront tous les NFT qu'ils possèdent depuis la chaîne de blocs.
- Les utilisateurs récupéreront ensuite toutes les métadonnées des NFT depuis l'API.

De plus, pour que chaque utilisateur puisse échanger et acheter des cartes, la DApp demande de la connexion au portefeuille MetaMask dès l'entrée sur le site.

2 Partie technique

2.1 Architecture du système

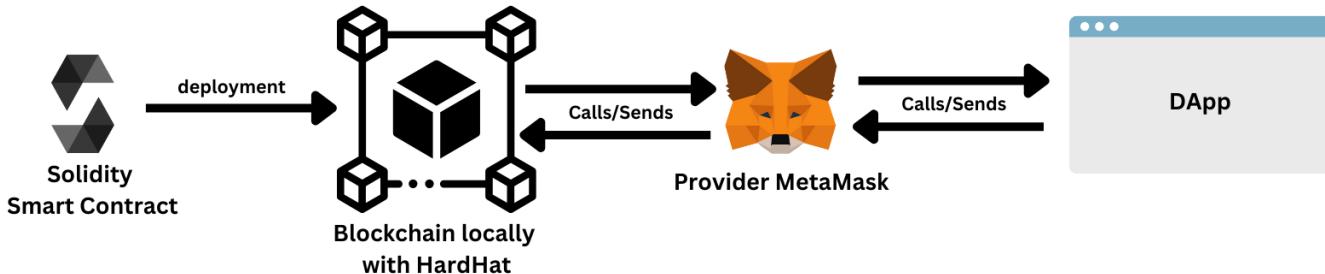


FIGURE 1 – Architecture de l’ensemble du système

Le Smart Contract une fois créé sera déployé en utilisant **HardHart**, ce dernier contrat n'est plus modifiable une fois sur la blockchain, ce qui permet de garantir une certaine sécurité (si les normes sont bien respectés) et de stocker de manière permanente les données. Toutefois, comme nous sommes à l'étape de développement, localement nous déployons notre Smart Contract à plusieurs reprises jusqu'à obtenir le contrat final.

Une fois le contrat présent sur la blockchain, nous récupérons son adresse et pouvons l'utiliser dans la DApp afin de faire des appels aux méthodes du contrat via des `call()` et `send()`, pour cela un provider comme MetaMask est nécessaire afin d'avoir accès à notre portefeuille.

2.2 Account

La page d'accueil du site correspond à l'Account. Pour afficher cette dernière il faut d'abord se connecter à MetaMask. Si l'extension n'est pas installée l'utilisateur se voit dirigé vers la page Install où se trouve un lien vers le site de Metamask pour l'installation.

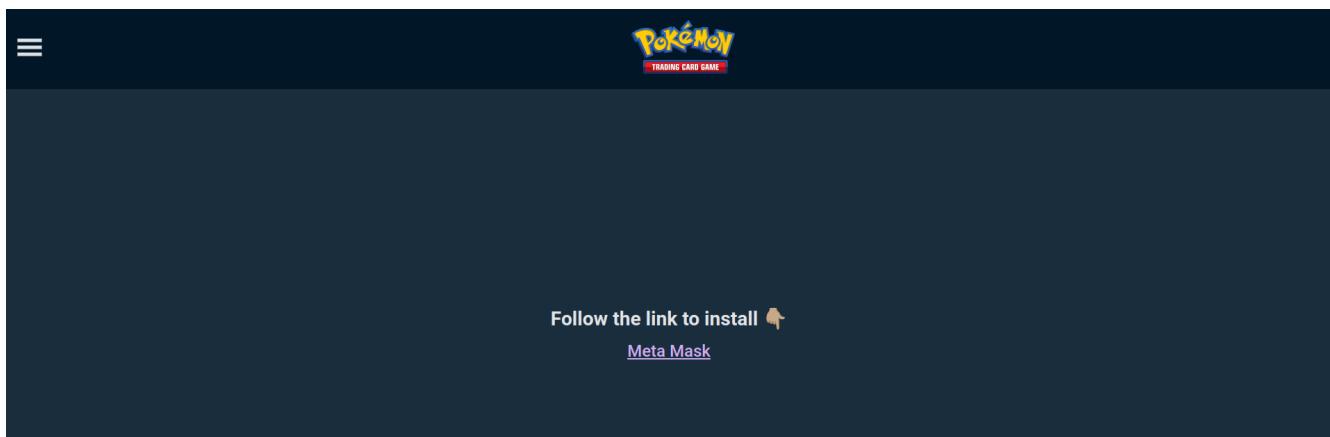


FIGURE 2 – Écran Install

Si l'utilisateur possède un portefeuille MetaMask, il sera directement dirigé vers l'écran Account qui lui permet de parcourir sa collection de cartes, si en possède une.

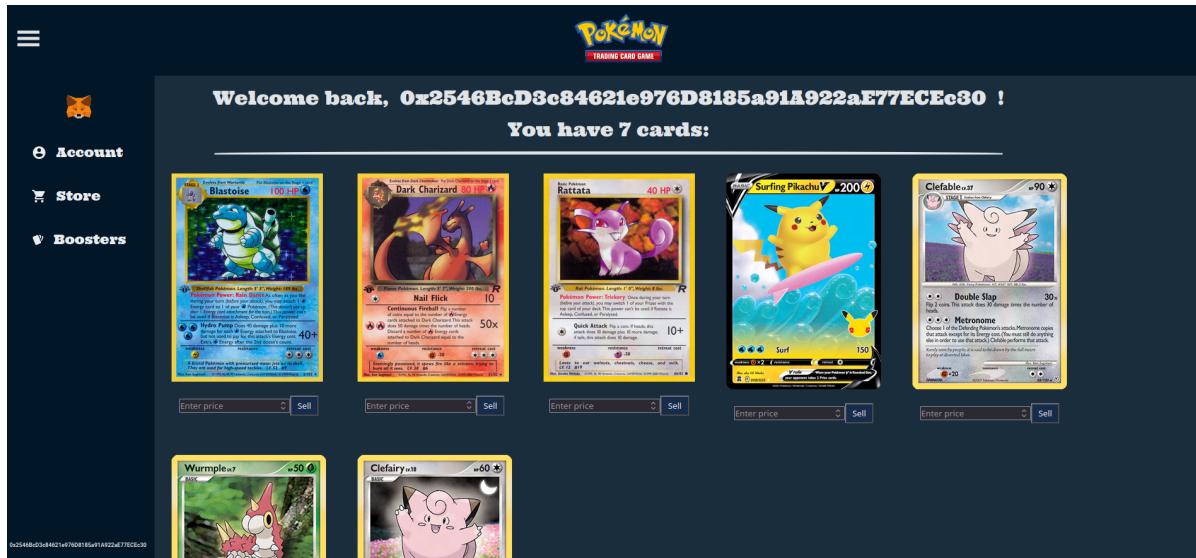


FIGURE 3 – Écran Account

A chaque fois que l'utilisateur achète depuis le marché ou gagne une carte dans un booster, elle est ajoutée à cette collection ou il pourra les mettre à la vente pour au moins 1 ETH s'il le souhaite.

2.3 Store

L'utilisateur peut donc acheter des cartes et pour cela il peut se rendre sur l'écran Store.

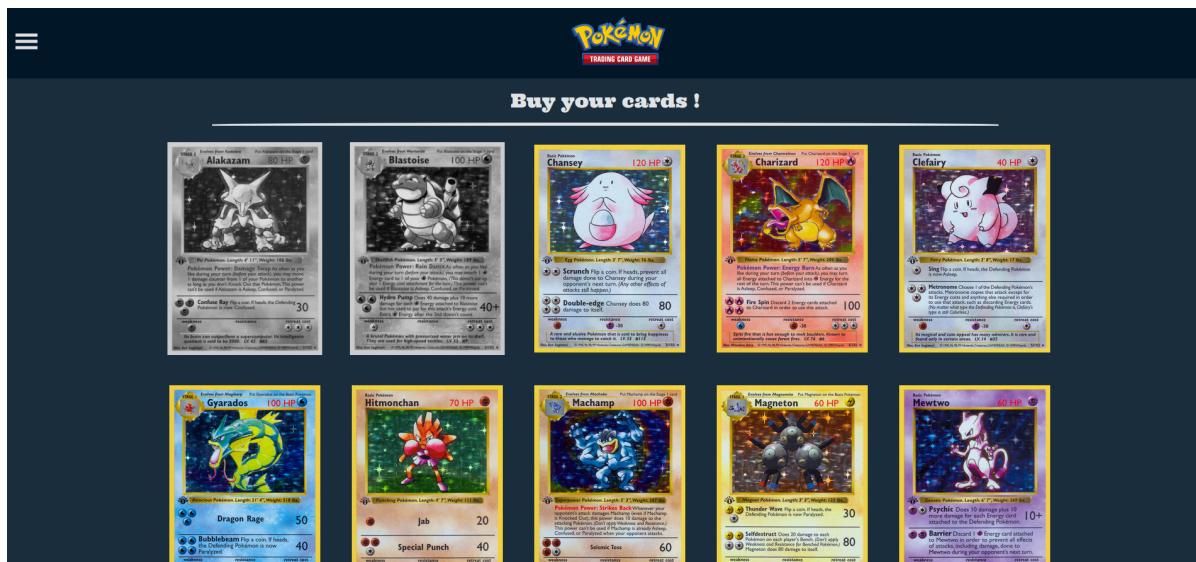


FIGURE 4 – Écran Store

Nous retrouvons toutes les cartes de la collection **Base** et aussi les cartes récupérées depuis les boosters et mises à la vente par les autres utilisateurs. Pour acheter une carte il suffit de cliquer dessus, s'il la carte s'agrandit et la souris se transforme en curseur alors la carte peut être acheté. La transaction s'affiche directement sur MetaMask avec le prix demandé pour la carte. Les cartes qui ne sont plus en vente s'agrandissent pas et sont grisées.

A noter : Il y a des cartes qui sont grisées et s'agrandissent (et la souris se transforme en curseur également), ce dernières sont celles mises en vente par les utilisateurs (généralement récupérés depuis les boosters).

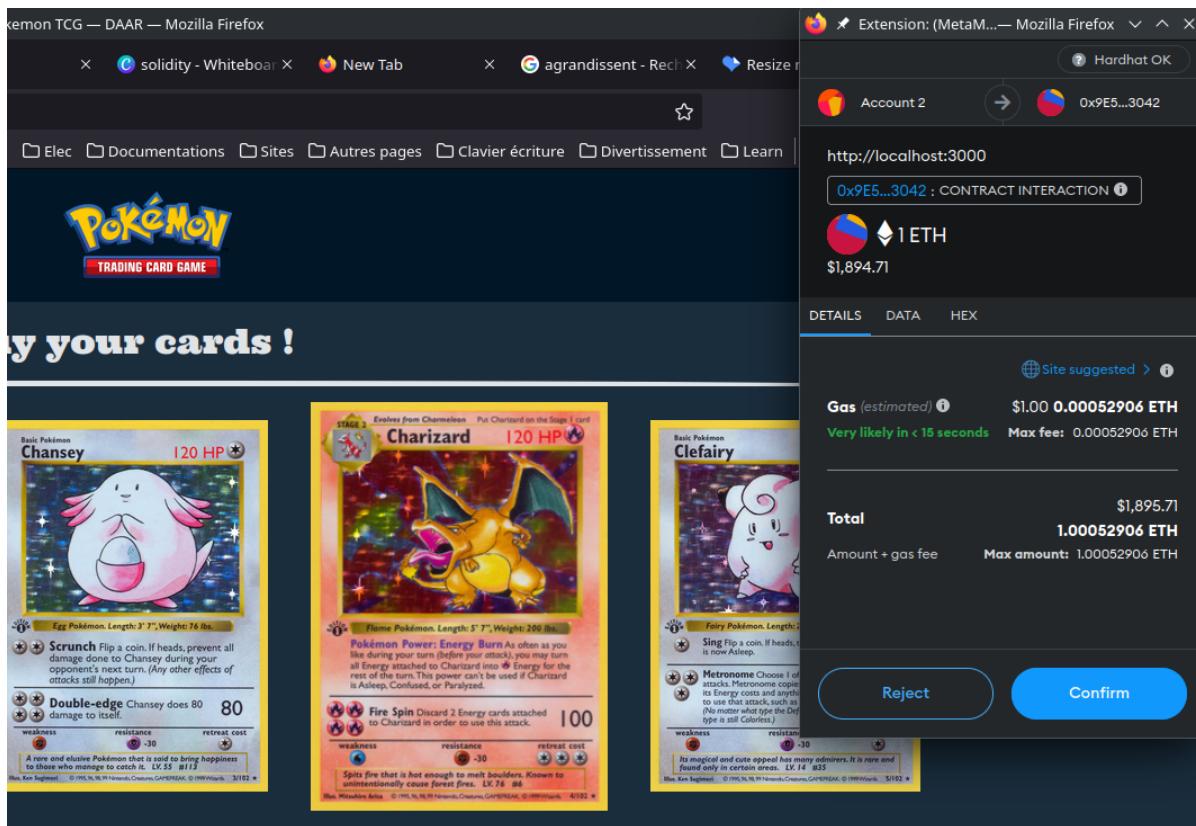


FIGURE 5 – Transaction sur MetaMask pour acheter une carte

La transaction se fait d'un compte à un autre sur la blockchain. L'utilisateur voit le coût de la transaction (ici 1 ETH) et également le coût estimé du Gas (dependant de la complexité de l'opération) qui s'additionne au prix de la carte. Il peut alors confirmer la transaction ou la rejeter.

2.4 Boosters

Finalement l'utilisateur peut également se procurer les cartes en achetant des boosters sur l'écran Boosters.

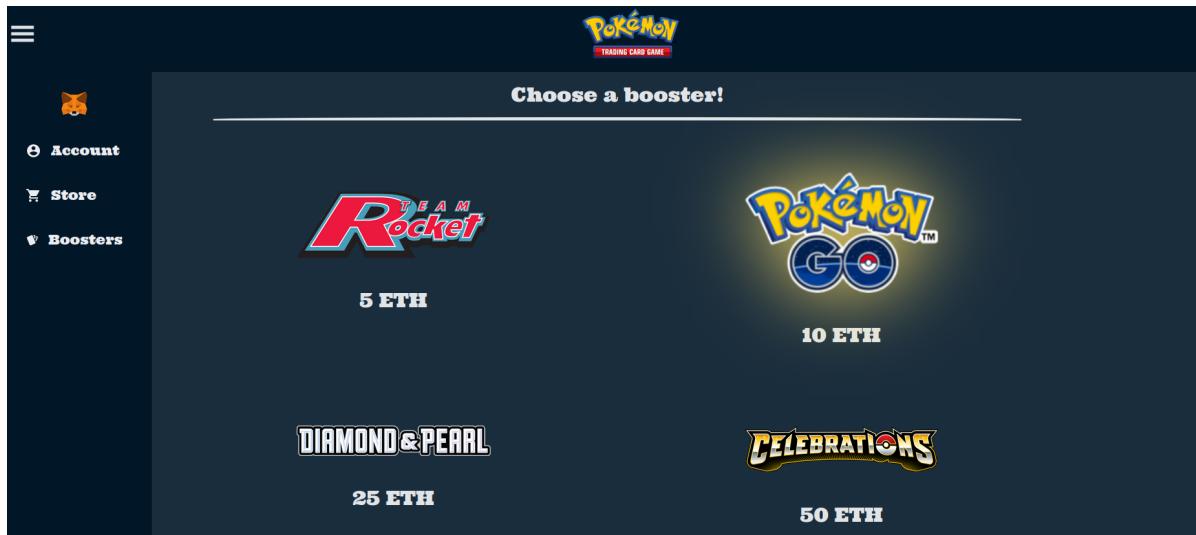


FIGURE 6 – Écran Boosters

Il a le choix entre plusieurs boosters. Chaque booster correspond à une collection (on pourra évidemment en inclure d'avantage par la suite) et chacun a un prix différent. De la même manière qu'avec la page **Store** l'utilisateur peut acheter un booster en cliquant dessus ce qui ouvre une transaction sur Metamask.

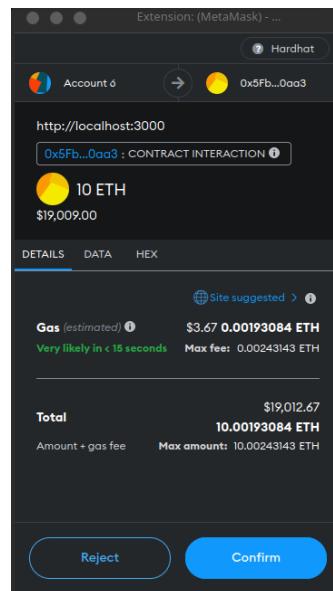


FIGURE 7 – Transaction pour le booster de la collection PokemonGo

L'utilisateur peut à nouveau confirmer ou rejeter la transaction. Si ce dernier confirme l'achat du booster, une animation d'ouverture se lance sur la page et les cartes s'affichent.

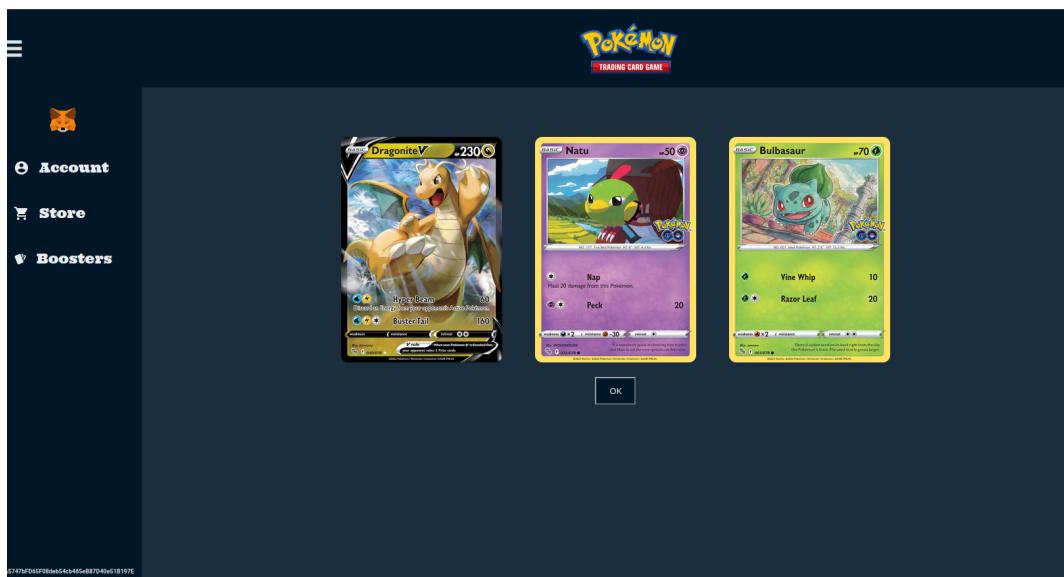


FIGURE 8 – Opening du booster

Les cartes gagnées sont alors directement ajoutées à la collection de l'utilisateur dans **Account**, qui pourra par la suite les revendre à nouveau pour le prix qu'il souhaite ou attendre que leurs valeurs augmentent d'avantage.

3 Tests de performance

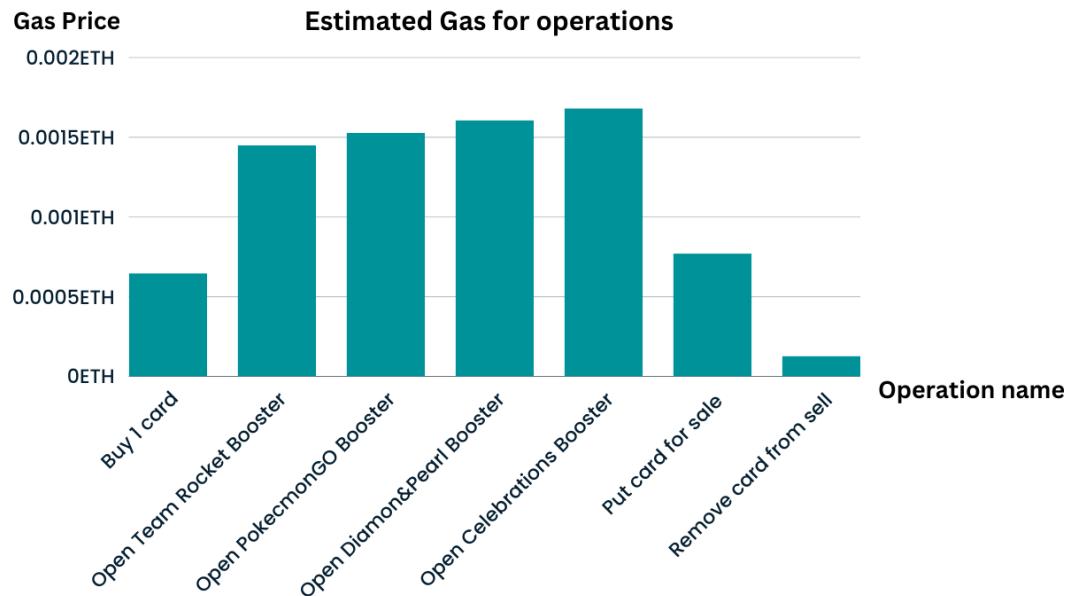


FIGURE 9 – "Gas" estimé des principales opérations

Une estimation du "Gas" pour les principales opérations s'avère intéressante, ce dernier est une mesure de coût et de complexité. Il permet de garantir que les utilisateurs ne surchargent pas le réseau, de la même manière cela optimise les ressources puisque les validateurs peuvent choisir quelle opération faire en fonction de cette mesure. Finalement, cette mesure du "Gas" joue un rôle crucial en renforçant la sécurité sur la blockchain.

Comme nous pouvons le voir dans l'histogramme ci-dessus, l'ouverture des boosters, qui sont en réalité l'achat de 3 cartes au même temps, coûte environ 3 fois plus en "gas" qu'en acheter qu'une seule. En retenant toutefois que pour obtenir ces 3 cartes il faut tirer au moins 3 nombres "au hasard" (au moins puisqu'il peut déjà exister des cartes avec ces numéros générés de manière pseudo-aléatoire) et manipuler des chaînes de caractères.

De l'autre côté, remettre une carte en vente coûte plus cher en "gas" que de l'enlever car pour l'ajouter nous avons choisi de parcourir toutes les cartes jusqu'à trouver la carte qui nous intéresse alors que pour l'enlever nous avons opté pour un mapping, en fournissant directement l'identifiant de la carte à supprimer (pour comparer le "gas" seulement).

4 Conclusion

La problématique de ce projet était de réaliser un jeu de cartes à collectionner sous forme de NFTs en utilisant la blockchain Ethereum et en développant le front-end avec React. Cet objectif a été atteint, nous avons créé une plateforme sur laquelle l'utilisateur peut acheter des cartes et ouvrir des boosters pour en gagner, de même il peut échanger en vendant les cartes pour amasser des ETH. L'intégration de transactions en utilisant le langage **Solidity** était complexe à mettre en place mais nous avons réussi à atteindre la majorité de nos objectifs en développant une plateforme fonctionnelle.

Références

- [1] CRYPTO ZOMBIES. Crypto Zombies. <https://cryptozombies.io/fr/>.
- [2] ETHEREUM ID. Remix. <https://remix.ethereum.org>.
- [3] POKEMONTCG. Pokémon TCG API. <https://pokemontcg.io/>.