TRY: a nfT lotteRY

Federico Bernacca

1 Introduzione

Per svolgere l'homework sono state effettuate le seguenti scelte progettuali.
I collectibles assegnabili agli NFT si trovano nella seguente repo: nft_lottery.
L'idea di base è che il *tokenId* dell'NFT è anche il nome del collectible a lui associato:

```
"_tokenId": "1",
"_image": "https://github.com/fedehsq/nft_lottery/master/collectibles/1.svg"
```

I contratti all'interno del progetto sono 3:

1. **ERC721**
2. **NFT**
3. **Lottery**

1.1 ERC721

La prima definizione è del contratto astratto **ERC721.sol**, che prevede la definizione dei metodi ed eventi necessari per essere conformi allo standard ERC721.

1.2 NFT

La seconda definizione ed implementazione è del contratto **NFT.sol** che eredita dal contratto sopra.

1.3 Lottery

Il terzo contratto prevede l'implementazione della lotteria, che utilizza il contratto **NFT.sol** per minare i token.

2 Funzionamento

Le funzionalità richieste sono state implementate correttamente, in più è stata fatta la seguente assunzione: *Se il numero di NFT disponibili per ogni classe durante l'estrazione dei primi è minore del numero di vincitori, viene minato un token on demand e inviato al vincitore.*

2.1 Log

Sono riportati i log delle operazioni richieste dal contratto tramite *Event* ed *Emit* di questi.

2.2 Operazioni

Per un corretto funzionamento del contratto l'ordine di deploy dei contratti e delle operazioni dovrebbe essere il seguente:

1. Deploy di **NFT.sol**.
2. Deploy di **Lottery.sol** con argomenti *(nftAddress, roundDuration i.e 2)*. Il primo round risulta aperto automaticamente dopo il deploy.
3. Minare N token (i.e 10) tramite la funzione *mintNtoken*.
4. Cambiare indirizzo del sender.
5. Acquistare *N* biglietti (i.e 10) tramite la funzione *buyNRandomTicket*.
6. Ripetere step 4 - 5.
7. Riportarsi sull'address relativo al *lotteryManager* .
8. Eseguire *drawNumbers* per estrarre i numeri vincenti.
9. Eseguire *givePrizes* per assegnare i premi.
10. Adesso è possibile aprire un nuovo round o chiudere la lotteria tramite le rispettive funzioni *openRound* e *closeLottery*.

Il consiglio è di utilizzare **ganache** piuttosto che **Js** perché questo crasha in continuazione.

2.3 Gas estimation

Per la stima del gas, l'idea è stata quella di trovare una funzione con un alto consumo di gas e cercare di migliorarla.
La funzione presa in esame è stata *givePrizes*.
L'implementazione iniziale prevedeva un ciclo in cui si confrontavano tutti i numeri di un biglietto con i numeri vincenti estratti, per una complessità di $O(n^2)$.
L'dea successiva ha previsto di ordinare i numeri in ordine crescente dei biglietti una volta comprati, per poi poter utilizzare una ricerca binaria in fase di confronto per ottenere una complessità totale di $n * log(n)$.
Le differenze in termini di gas speso sono simili per l'esperimento condotto:

```
nTicket: 200
binarySearch gas cost: 21782135
normalSearch gas cost: 21561213
```

2.4 Security

Il generatore random risulta sicuro perché facendo l'operazione **blockhash** di un blocco $(X + K)$ che ancora non è sulla blockchain al termine del round, ovvero al termine dell'operazione di acquisto dei biglietti, un attaccante non può predire i numeri estratti dal generatore perché durante il round, questo blocco non è stato ancora minato, quindi risulta impossibile prevedere il suo hash.
(Per semplicità il parametro K è settato a 0, poiché essendo locale la blockchain, si dovrebbero fare K ulteriori transazioni prima di estrarre i numeri vincenti).