# TRY

Antonio Pace

May 2022

## 1 TRY

For the implementation of TRY, I created two smart contracts: one for the NFT and one for the lottery.

### 1.1 Cryptoducks.sol

The NFT contract is written starting from `https://github.com/nibbstack/erc721/src/contracts/tokens/nf-token-metadata.sol` that is an implementation of ERC-721 standard, with methods for mint, transfer, set metadata and check the ownership of NFTs.

*Cryptoducks.sol* is a subclass of this implementation, with only two methods: one for minting the NFT linking a metadata to it, and another for transfer the NFT from one account to another one, checking that the operation is allowed.

### 1.2 lottery.sol

*lottery.sol* is the implementation of the lottery logic. It contains all the methods required by the assignment, plus some auxiliary methods:

- **random(int range, int seed)**: (*private*) generate a random number from 1 to *range*.
- **checkDuplicates(int a, int[] b, int i)**: (*pure, private*) checks if integer *a* is present in the array *b[1..i]*. I need it for checking that the random winning numbers generated are all different.
- **findCommonElements(int [] c, int [] d)**: (*pure, private*) finds common elements between arrays *c* and *d*.

- **withdraw(address addr)**: (*public*), withdraws the revenue of the lottery sending it to *addr*. Can be called only by the lottery manager.
- **mintOnDemand(address to, uint tokenId, string uri)**: (*private*) for minting an NFT when necessary, i.e if the NFTs minted at the beginning of the lottery are already assigned. The difference from **mintDKS()** is that here the token ids are not stored for later assignment but just minted on the fly.
- **transferDKS(address to, uint tokenId)**: (*private*) for transfer an NFT from one address (the manager that minted the NFTs at the beginning) to another one.

## 1.3 Implementative choices

Players' address and tickets are stored in an array of *struct Player*, that is cleared every end of a round.

For the NFTs, there is a mapping between the classes and the NFTs (*struct DKStoken*) minted at the beginning of the lottery, that gives the informations about the ID, the URI and if the NFT is already assigned. This is important for decide if transfer the existing NFT or minting a new one on the fly, and for knowing the metadata URI for every class.

There is a variable for counting the ID of the tokens that have to be minted on the fly. So it's very important that the lottery manager mints the NFTs always progressing in an increasing way.

The length of each round is set at the deployment of the lottery contract.

When the lottery is closed, players are refunded if necessary (round not finished), and a boolean variable is set at false (*lotteryUp*), blocking the possibility to start a new round of the lottery.

A round should be active for M blocks. I check for this in the **buy(...)** function, deactivating the round if M blocks are passed. I tested the application in local, so there is one block for every transaction and there are only blocks relative to the lottery. In a deployment over a blockchain, there are many different blocks, so it can happens that the function doesn't detect immediately the M blocks passed. Moreover, if nobody buys the ticket, the check doesn't take place.

## 1.4 Execution of TRY

First of all, the deployment of the two contract and the minting should be done with the same address (the lottery manager). I will present some steps for a simulation of the lottery:

- Deployment of Cryptoducks.sol.
- Deployment of lottery.sol, the constructor take as arguments the parameter M (duration of the lottery in blocks) and the address of Cryptoducks contract (for calling mint and transfer).
- The lottery manager mints 8 NFT to himself (**mintDKS(...)**), one for each class, with tokenIds from 1 to 8, assigning the URI of the Cryptoducks' images according to Figure 1.
- The lottery manager calls **startNewRound()**. The round is now active.
- Now, players can buy a ticket for 610'000 gwei (as tx value) calling **buy(...)**, choosing 5 different numbers from 1 to 69 and one powerball from 1 to 26 (the last one). An event shows the ticket.
- When M blocks are passed, the round is not active anymore, and the lottery manager can call **drawNumbers()**. An event shows the 6 winning numbers.
- The lottery manager can now call **givePrizes()** for distribute the prizes to the winning players, according to the guessed numbers. Events show the transfer of the NFT (from, to, tokenId) and the won prize (class, uri). The first time that someone win a prize of a class *c*, receives the one previously minted. Then following prizes are minted on the fly. The round is now finished and the data structure containing the players is cleared.
- The lottery manager can now starts a new round.
- The lottery manager can **withdraw(...)** the revenues of the tickets on an address of his choice.
- The lottery manager can close the lottery when he want. If the round is not finished, players are refunded.

## 1.5 Gas

Surely, the transaction that cost more gas is **givePrizes()** if the tickets are more than a couple. That's because the function has to check for every ticket, how many number the player has guessed, if he guessed the powerball and which prize should be assigned, and that means nested loops. So for a big number of tickets the gas price is high, while the other function's cost are almost constant. With 10 tickets, Remix says that the function call costs 420787 gas, that is almost the double of the **drawNumbers()** cost (212863 gas), which has to generate 6 random numbers, checking that they are different.

## 1.6 RNG

For extracting the winning numbers, there is a need for a random sequence. I used an hash of the timestamp of the block with an index for changing the sequence. Due to the deterministic nature of the method, this is a pseudo-random generator. For a true random number generator we may need an oracle. In fact, knowing these information, everyone can reconstruct the sequence. However, the winning numbers are generated when is not possible anymore to buy the tickets, so it's impossible to predict in advace the right sequence. The real problem can be the manipulation of the block by miners, but this is an issue only if the prize of the lottery is valuable enough.

# 2 Cryptoducks

Cryptoducks are a batch of collectibles, designed specifically for this assignment (special thanks to Lisa Queirolo). In Figure 1, you can see a Cryptoduck for every class. These collectibles are intended to be uploaded on IPFS or a standard image-hosting website and set as metadata for the NFT prizes of TRY.

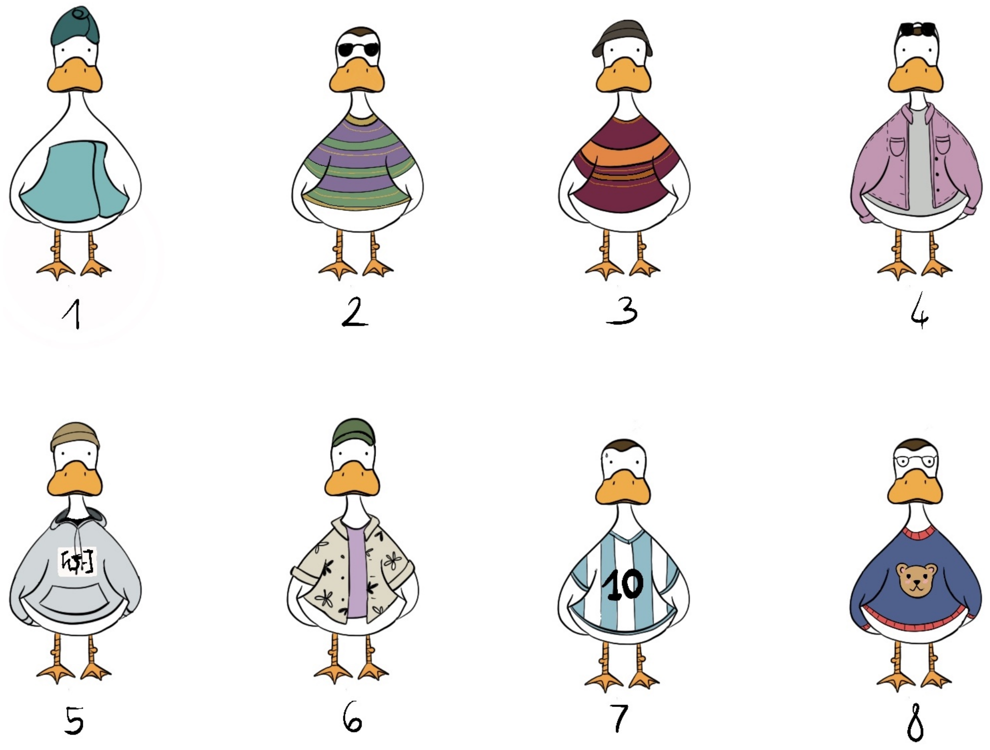

Figure 1: Cryptoducks