Fintech and Cryptocurrencies

Coding Week 2020

As you have probably already noticed, Co is obsessed with sneakers. Since he wears them every day, he is certain he knows what good sneakers are. This is why he designed his own pair:

Each pair is personalized, the buyer can choose the name at the back and the image on the side (of course, Co chose the unicorn). Also, the public key of the buyer is engraved into the sole.

Question 1 [40p]

Since all of these pairs of shoes are unique and a piece of art, why not create a smart contract to keep a digital twin of them?

Tasks

1. Create a truffle project
2. Write a smart contract called `CoShoe` that holds non-fungible tokens:
    a. Each shoe is a struct called `Shoe` comprised of:
        i. `owner` (address)
        ii. `name` (string)
        iii. `image` (string: url to the image)
        iv. `sold` (bool)
    b. Define a state variable called `price` and set it to 0.5 Ether, converted to Wei.

    c. Define a state variable called `shoesSold` that holds the number of shoes that have already been sold. Set it to 0.
    d. Define a public array called `shoes` that holds instances of `Shoe`
    e. Implement a `constructor` that mints 100 `CoShoe` tokens. The `owner` of each token is the address deploying the contract, `name` and `image` are empty strings (`""`), and `sold` is equal to `false`. Add the instances of `Shoe` to the array `shoes`.
    f. Implement a function called `buyShoe` that
        i. Takes the input parameters `name`, `image`
        ii. Checks that there is still a pair of shoes left that has not been sold yet, otherwise it throws an error
        iii. Checks that the value that is attached to the function call equal the `price`, otherwise it throws an error
        iv. Transfers the ownership of a `Shoe` to the caller of the function by setting `owner` within the `Shoe` struct, setting `name` and `image` to the input variables, and changing `sold` to `true`
        v. Don't forget to update `soldShoes`
    g. Implement a function called `checkPurchases` that
        i. returns an array of bools that are set to true if the equivalent index in `shoes` belongs to caller of this function
        Example: `[true, false, false, false, false, true, false, false, …]`
        ii. Remember to implement it in a gas saving manor
3. Test the following functionalities:
    a. 100 tokens are minted on deployment
    b. `buyShoe` correctly transfers ownership, sets the name and the image, sets sold, and updates `soldShoes` count
    c. `buyShoe` reverts if the price is not equal to 0.5 ether
    d. `checkPurchases` returns the correct number of `true`s
4. Compile your contract
5. Include a `2_deploy_contract.js` in the migrations folder
6. Deploy your contract to a network of your choice (this can be an emulated network)
7. Push your code to a new github repository
8. In addition to submitting on github, please hand in a complete zip file including the truffle project.

Question 2 [30p]

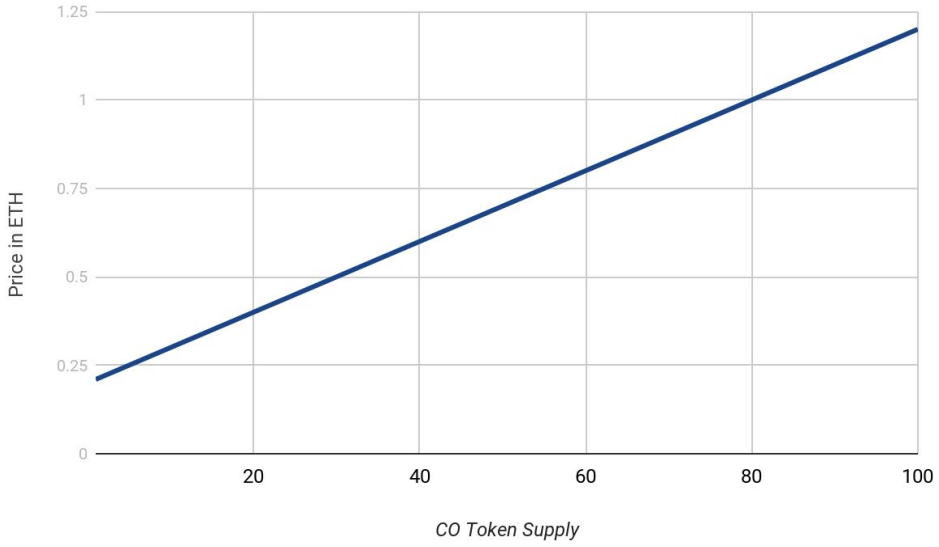
Instead of starting a kickstarter campaign, Co wants to use a token bonding curve to sell his shoes. Token bonding curves issue tokens through a buy and sell function.

Getting to know bonding curves:

- Introductory Article
- Presentation
- Token Bonding Curve Cheat Sheet
- Interview with the inventor

Here are Co's bonding curve specs:

The Token
Issuance: A fungible token called "CO"
Supply: 100, this is the number of shoes Co wants to produce
The Bond
Collateral: ETH, CO tokens are bought in ETH
The Traded Asset or Objective: The shoes, 1 pair cost one "CO"
The Curve



Curve function: $f(x) = 0.01x + 0.2$, $x \in \mathbb{N}$
Pricing: static pricing
Specialty
Only Co is allowed to sell tokens back to the curve.

Tasks

1. Create a truffle project
2. Write a smart contract to issue CO tokens:
    a. Make the contract “ownable”
    b. Implement the ERC20 functionalities
    c. Implement a `buyPrice` function that calculates the price for the purchase of `n` CO tokens based on the curve defined above.
    d. Implement a `sellPrice` function that calculates the price for the sale of `n` CO tokens based on the curve defined above.
    e. Implement a `mint` function that creates tokens if the correct current price is transferred to the contract. The price is determined by the `buyPrice` function.
    f. Implement a `burn` function that can only be called by the `owner` (i.e., only the `owner` can sell tokens back to the curve and withdraw the funds). The price is determined by the `sellPrice` function.
    g. Implement a `destroy` function that destructs the contract (see `selfdestruct`). This function can only be called by the `owner` and it can only be called if all CO tokens belong to the `owner`.
3. Test the following functions:
    a. `mint`
    b. `burn`
    c. `destroy`
4. Compile your contract
5. Include a `2_deploy_contract.js` in the migrations folder
6. Deploy your contract to a network of your choice (this can be an emulated network)
7. Push your code to a new github repository
8. In addition to submitting on github, please hand in a complete zip file including the truffle project.

Bonus Question 3 [10p]

Update your contracts from Question 1 (CoShoe) and 2 (CoToken) such that they interact with each other. For this, move them into a new truffe project, i.e. a truffle project that includes both contracts.

Updates in CoShoe:

1. `CoShoe` needs to know about `CoToken`
2. The `price` does not have to be defined because it is 1 CO.
3. The `buyShoe` function does not accept Ether. Instead, it checks on the `CoToken` contract if the address calling this function owns a CO token.
4. If the address calling the `buyShoe` function owns a CO token, it triggers the `transferFrom` function in the `CoToken` contract and transfers the token to the `owner`

of the `CoToken` contract. Make sure that the function call reverts if the CoTokens could not be transferred.

Updates in CoToken:

1. You may have to make the `owner` public in order to retrieve the recipient address for the `transferFrom` function.

Deploy the updated version of your contracts to a network of your choice. Push your code to a new github repository. In addition to submitting on github, please hand in a complete zip file including the truffle project.

Deadline: Sunday, 08/03/2020, 18:00
Please send an email including your submission to brtsab001@myuct.ac.za