Bull and Bear Exchange

DMBLOCK Assignment 2

Lukáš Častven, Jakub Jelínek
Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
xcastven@stuba.sk, xjelinekj@stuba.sk

April 28, 2024

Contents

1 Assignment

Main goal of this assignment was to complete provided implementation of an Uniswap [1] inspired decentralized exchange. We were given solidity and javascript source codes in a Hardhat project, and we had to implement methods in these files to create a functional decentralized exchange for swapping Ether with our custom ERC20 token.

2 Questions

1. Why removing liquidity from exchange doesn't change the rate

   When a liquidity provider adds liquidity, the provided amount of Eth and BBT must have the same value (according to the current exchange rate) in order to preserve the exchange rate of those two assets.

   For example, if exchange rate is 4 Eth for 3 BBT (4:3), provider must provide $4n$ Eth and $3n$ BBT, where $n \in \mathbb{N}^+$

2. Explain implemented fee mechanism for incentivizing liquidity providers

   During each swap, user's input amount of Eth or BBT is converted to amount of the other asset with equivalent value, and this amount would be swapped to the user. However, when a fee of value $f\%$ is present, this converted is reduced by the fee, which stays in the pool and user gets $(1 - f) * converted\_amount$.

   Example, user wants to swap 100 Eth for BBT, or vice versa, at exchange rate 1:1, and $x = 5000, y = 5000 \Rightarrow k = 25000$. All calculations are in integer format, meaning no floating numbers. The move on the constant product formula tells us that the amount to be swapped is:

$$(x + dx) * (y - dy) = k$$

$$(5000 + 100) * (5000 - dy) = 25000$$

$$dy \approx 98$$

   98 is the amount without fees. To apply a fee of $f = 3\%$, it must be converted to a fractional form of $f = \frac{fee\_numerator}{fee\_denominator}$ and then applied like this:

$$\begin{aligned} with\_fee &= 98 * (1 - f) \\ &= 98 * (1 - \frac{fee\_numerator}{fee\_denominator}) \\ &= 98 * \frac{fee\_denominator - fee\_numerator}{fee\_denominator} \\ &= 98 * \frac{100 - 3}{100} \\ &\approx 95 \end{aligned} \tag{1}$$

So the final amount of BBT (or Eth) to be swapped at given exchange state is 95, meaning 3 units of swapped asset stay in the pool as a fee.

When a liquidity provider deposits liquidity into a pool, he/she owns a portion of the pool, and based on that portion, fees will be payed back when the liquidity will be removed. The portion is tracked in the contract for each provider, and when new liquidity is added, the amount of Eth deposited is added to the provider's liquidity portion. Then when withdrawing liquidity, providers gets:

- Removed liquidity amount of Eth,
- Amount of BBT equivalent in value to the Eth amount,
- And additional assets according to the provider's portion of the pool, where the portion is calculated as $\frac{providers\_liquidity}{total\_liquidity}$.

3. Explain at least one gas optimisation method you used

One method we used is called **CEI - checks, effects, interactions** [2]. **CEI** guides developer to first check for all possible preconditions that must be true to achieve desired functionality of the contract. Then the effects take place. These mutate the state of the contract. And interactions with external contracts are last.

The **effects** phase saves gas, because instead of reverting in the middle of a computation due to some false condition, the transaction reverts at the beginning and less computation is done, meaning less gas is spent.

The last phase also helps with reentrancy bugs, but that is out of scope for this answer.

Feedback questions

4. How much time did you spend on the assignment

Each of us spent around 25 hours on this assignment.

5. What would be one useful information before you started to work on this assignment

The switch in mentality from having a standard frontend-backend application to frontend-blockchain would help in the beginning.

6. What one thing you would change

Some automated test suite from you would be nice, for example, implement this interface on contract and here is a Hardhat/Foundry/... test file which gives you confidence that what you implemented is correct.

3 Implementation

We decided to rewrite the provided Hardhat project into Foundry [3]. Our decentralized exchange is called **Bull & Bear Exchange** and the ERC20 token traded on this exchange is **Bull & Bear Token**. Also we rewrote provided web app into Vue [4].

The project structure looks like this:

- `app` - contains the Vue frontend interacting with the smart contracts
- `dex` - contains the Foundry project for the smart contracts of **Bull & Bear Exchange**
- `docs` - contains documentation for this assignment

Bull & Bear Token — BBT

ERC20 token to be traded on our exchange is called **Bull & Bear token**, with symbol being **BBT**. We argue that the two functions from assignment (`mint` and `disable_mint`), which we have to implement, are useless and potentially an anti-pattern. ERC20 implementation by OpenZeppelin is enough to implement a token with constant supply. By pre-minting supply to the deployer of the token, we achieved a token with constant supply (no new tokens can be minted as `_mint` in ERC20 is an internal function [5]). Thus we have reduced the complexity of this token implementation by removing `mint`, `disable_mint` and even the `Ownable` parent contract used in provided source code.

Thus the whole contract has few lines and minimal complexity:

```
import {ERC20} from "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract BBToken is ERC20 {
    constructor(uint256 supply) ERC20("Bull and Bear Token", "BBT") {
        _mint(msg.sender, supply * 10 ** decimals());
    }

    function decimals() public pure override returns (uint8) {
        return 0;
    }
}
```

The assignment requires that our token be indivisible, the function decimals is overridden to reflect this requirement.

Bull & Bear Exchange

Swapping

The swapping functions are fairly straight forward and look similar. First check conditions needed for the swap, calculate amount to be swapped back to user, check that it didn't slip out of user tolerable range, update reserves and send the asset to user.

```
function swapETHForTokens(uint256 minTokenAmount) external payable {
    uint256 weiAmount = msg.value;
    require(weiAmount > 0, "Need ETH to swap");

    (uint256 tokenAmount, uint256 withFee) = getTokenAmount(weiAmount);
    require(tokenReserves - tokenAmount > MIN_LIQUIDITY, "Not enough liquidity");
    require(withFee >= minTokenAmount, "Too much slippage");

    weiReserves += weiAmount;
    tokenReserves -= withFee;

    token.transfer(msg.sender, withFee);
}

function swapTokensForETH(uint256 tokenAmount, uint256 minWeiAmount) external {
    require(tokenAmount > 0, "Need tokens to swap");
    require(token.balanceOf(msg.sender) >= tokenAmount, "Not enough tokens");
    require(
        token.allowance(msg.sender, address(this)) >= tokenAmount,
        "Not enough token allowed for transfer"
    );

    (uint256 weiAmount, uint256 withFee) = getWeiAmount(tokenAmount);
    require(weiReserves - weiAmount > MIN_LIQUIDITY, "Not enough liquidity");
    require(withFee >= minWeiAmount, "Too much slippage");

    token.transferFrom(msg.sender, address(this), tokenAmount);

    tokenReserves += tokenAmount;
    weiReserves -= withFee;

    (bool succes,) = payable(msg.sender).call{value: withFee}("");
    require(succes, "Transfer failed");
}
```

Liquidity

Liquidity is tracked by how much provider deposited Eth to the pool. Then on removal he/she gets the same amount of Eth as liquidity being removed and amount of token in equivalent value to the Eth amount.

Web app

We also rewrote provided jQuery web application with VueJS. To start the web app Foundry and Node +20 (or Bun) are needed, then navigate to the *app* directory and run:

```
npm i; npm run dev
```

Then navigate to *dex* and run:

```
forge script ./script/DeployContracts.s.sol \
    --rpc-url 'http://localhost:8545' --broadcast -vvv
```

The web app will be available at `localhost:5173`.

4 Testing

We didn't rewrite sanity check from the provided source codes, instead we used Foundry and wrote a lot of tests. The coverage report generated by Foundry looks like this:

File	% Lines	% Statements	% Branches	% Funcs
script/DeployConfig.s.sol	0.00% (0/2)	0.00% (0/4)	100.00% (0/0)	0.00% (0/2)
script/DeployContracts.s.sol	94.12% (16/17)	95.24% (20/21)	100.00% (0/0)	50.00% (2/4)
src/BBExchange.sol	95.06% (77/81)	95.74% (90/94)	88.57% (62/70)	83.33% (10/12)
src/BBLibrary.sol	100.00% (8/8)	100.00% (19/19)	100.00% (0/0)	100.00% (3/3)
src/BBToken.sol	0.00% (0/1)	0.00% (0/1)	100.00% (0/0)	100.00% (1/1)
Total	92.66% (101/109)	92.81% (129/139)	88.57% (62/70)	72.73% (16/22)

Table 1: Code Coverage Metrics

5 Security analysis

For static analysis we used Slither [6]. A report can be found in the *docs/slither-report.txt*. We applied first major findings, which were about ingoring return values from `transferFrom` calls. There were some false positives in the findings, for example the finding about strict equality is one. Even though there are few reentrancy findings, we believe these are also a false positives.

6 Our improvements

We have made two improvements to the original implementation of the decentralized exchange. First of all, we have used Foundry instead of Hardhat. We have also rewritten the web application in Vue.js, which is a more modern and user-friendly framework than the original jQuery-based "application".

7 Conclusion

In this assignment, we have learned how to create and implement a decentralized exchange inspired by Uniswap v1. Thanks to that, we were able to implement smart contracts for the exchange and the ERC20 token, using Solidity and Foundry. These contracts were then connected to a web application rewritten in Vue.js, which allows users to interact with the exchange.

References

[1] "Overview — Uniswap — docs.uniswap.org." https://docs.uniswap.org/contracts/v3/overview. [Accessed 27-04-2024].

[2] "Security Considerations; Solidity 0.8.25 documentation — docs.soliditylang.org." https://docs.soliditylang.org/en/v0.8.25/security-considerations.html#use-the-checks-effects-interactions-pattern. [Accessed 28-04-2024].

[3] "Foundry Book — book.getfoundry.sh." https://book.getfoundry.sh/. [Accessed 27-04-2024].

[4] "Vue.js — vuejs.org." https://vuejs.org/. [Accessed 27-04-2024].

[5] "ERC 20 - OpenZeppelin Docs — docs.openzeppelin.com." https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#ERC20-_mint-address-uint256-. [Accessed 27-04-2024].

[6] "GitHub - crytic/slither: Static Analyzer for Solidity and Vyper — github.com." https://github.com/crytic/slither. [Accessed 28-04-2024].