CS6858: Module 3 Project Report

Decentralized Exchange

Prasanna Bartakke EE19B106

Introduction

Uniswap is a decentralized exchange(DEX), an alternative to traditional centralized exchanges. Recently, we have seen the collapse of big traditional centralized cryptocurrency exchanges like FTX. Uniswap is decentralized and highly safe, and as the liquidity pool is built on Ethereum, it has the same security as the Ethereum blockchain.

Market makers provide liquidity to markets. In traditional exchanges, trades happen using an order book. Liquidity makes transactions possible: if one wants to sell something at a particular price, but no one is buying it at that price, the trade will not happen.

The main challenge in building decentralized exchanges is maintaining enough liquidity. The best way to do this is to allow anyone to become a market maker.

Building a DEX similar to Uniswap is an essential step toward the DeFi economy. In this project, I thoroughly studied Uniswap's protocol from the whitepaper. I have implemented a decentralized exchange that enables anyone to become a liquidity provider. Users can trade tokens listed on the exchange for ETH or vice-versa. Users can also directly swap tokens for some other kinds of tokens. Some of the key challenges that I faced during the implementation of the DEX are as follows:

1. Developing a gas-efficient implementation of the reward mechanism for the liquidity providers.
2. Making the trades resilient to sandwich attacks from adversaries.
3. Lack of support for float type in Solidity.
4. Maintaining the security of each function by adding appropriate require statements.

How DEX works?

A decentralized exchange consists of two types of participants: liquidity providers (LP) and traders. Liquidity providers provide equal value for both types of liquidity (ETH and the token). When traders exchange one currency for another, they will add one currency to the liquidity pool and receive an equivalent value of the other currency from the pool. The constant product formula determines the exchange rate between the two currencies.

Let x be the amount of currency A in the liquidity pool, and let y be the amount of currency B. Let k be a constant called the exchange invariant. After every swap, the following condition must be satisfied: $x * y = k$.

The exchange sends the appropriate amount of the swapped-out currency during each swap. The exchange invariant is kept constant after each swap. However, this is only

partially true, as due to the exchange fees, the exchange invariant grows with each trade. This is explained in detail later.

Suppose a trader wants to trade Δx amount of A, for Δy amount of B. We can calculate Δy, the amount of B that the exchange will return to the trader as follows(assuming that the exchange fees are 0 for now):

$$(x + \Delta x)(y - \Delta y) = xy$$

$$\Delta y = \frac{y\Delta x}{(x + \Delta x)}$$

The price of currency B in terms of currency A can be calculated as x/y, whereas the price of currency A in terms of currency B can be calculated as y/x. Every swap modifies the exchange rate, and it is an indication of demand for a given currency.

Adding liquidity does not change the price, as the amounts of A and B added to the reserves are proportional to their value. The exchange invariant increases as liquidity increases.

Rewarding Liquidity Providers

We need to reward the liquidity providers for providing liquidity to the exchange. The easiest way is to charge a small fee(0.25%) for each swap by the users and add it to the liquidity pool. Thus, this is the reason why the exchange invariant increases with time.

The liquidity providers can remove liquidity at any time and withdraw an equal value of each currency. They are not entitled to withdraw their exact initial investment (in terms of the absolute quantity of each token). Instead, providing liquidity is similar to owning a percentage share of the liquidity pool, which the liquidity provider can withdraw later. A liquidity provider who provided x% currency A and B is authorized to remove x% of each reserve for those currencies (assuming their percentage was not diluted by other providers). When new liquidity providers add liquidity, the exchange must update the percentage share of the old liquidity providers. A brute-force way to do this is to maintain a mapping from the address of the liquidity providers to their percentage share. The problem with this approach is that the amount of gas required will grow linearly with the number of liquidity providers, and our exchange will only work on a small scale.

The most elegant way to solve this problem is to provide LP tokens to the liquidity providers, indicating their share percentage in the exchange. LP tokens are ERC20 tokens issued to liquidity providers in exchange for their liquidity. LP tokens represent the share of a liquidity provider in the liquidity pool. The amount of tokens one gets is proportional to the percentage of one's liquidity in the pool's reserves. Fees are distributed proportionally to the amount of tokens one holds. LP tokens are exchanged back for liquidity + accumulated fees.

The LP tokens do not have a supply limit, and the exchange mints new LP tokens when new liquidity is added. The number of LP tokens minted when a new liquidity provider provides liquidity is amountMinted=totalAmount*(ethDeposited/ethReserve). Let's try to understand how the amount of LP tokens indicates the percentage of shares. When the first liquidity provider, LP0, provides liquidity(deposits x ETHs), he gets x LP tokens. He owns 100% of LP tokens and has a 100% share of the liquidity pool. Suppose a new liquidity provider, LP1, adds liquidity by depositing 2x ETHs and the equivalent amount of tokens. The amount of LP tokens minted equals x*(2x/x) = 2x. We can see that LP1 provided twice the amount of liquidity compared to LP0, and each liquidity provider's number of LP tokens indicates the percentage share in the liquidity pool. Percentage share of LP0 = x/3x = 33.33% and percentage share of LP1 = 2x/3x = 66.66%. When a liquidity provider removes liquidity, he tells the amount of LP tokens(percentage of share) he want to remove from the liquidity pool. If the amount of LP tokens is valid, those many LP tokens are burned, effectively adjusting the percentage share of other liquidity providers, and the amount of ETH and tokens corresponding to the input LP tokens are returned to him. Thus using LP tokens, we can dynamically assign the percentage share to each liquidity provider without using much gas.

Handling Slippage

Many users may be trying to trade tokens at once on a DEX. There can be a difference in the exchange rate between the submission of a trade and the actual addition of that transaction on the blockchain. This shift in the exchange rate between the exchange's quote price and the actual price is called slippage. Attackers can make use of this to create sandwich attacks. A sandwich attack works as follows:

1. Alice submits a swap transaction to convert some large amount of token A into token B.
2. An adversary sees Alice’s transaction and front-runs it with a very large purchase of currency B, thus raising the price of asset B.
3. Alice buys currency B at the new higher price, further raising the price of currency B.
4. The attacker then immediately sells all their newly acquired currency B at the higher price, making a quick profit.

Sandwich attacks are bad for users of a decentralized exchange, as users pay higher exchange rates than the true asset value. Sandwich attacks are avoided by allowing users to set some maximum slippage while submitting the transaction.

Types of swaps

1. ETH to Token: Swap the given amount of ETH for the equivalent value in exchange token and update the exchange state accordingly.

2. Token to ETH: Swap the given amount of tokens for the equivalent value of ETH and update the exchange state accordingly.
3. Token to Token: Start with the standard token-to-ether swap. Instead of sending ethers to the user, find the exchange for the token address provided by user. Send the ethers to that exchange to swap them to tokens. Return the swapped tokens to user.

The implementation details and tests are explained in the video.