

한 번에 끝내는 블록체인 개발 A to Z

Chapter 3

Defi 기초 컨셉 구현

Chapter 3

Defi 기초 컨셉 구현

유니스왑v1 CPMM 구현(1)

Goal

CPMM 기반의 AMM을 구현한다.

가격 측정 함수를 개선해본다.

CPMM

1. $xy = k$ 공식에서 스왑 후에도 k 는 변하지 않는 다는 것이다.
2. 스왑 후 xy 는 $(x + \Delta x)(y - \Delta y)$ 가 될 것이다. Δx 는 Input 토큰의 개수, Δy 는 Output 토큰 개수이다.
3. 따라서 $(x + \Delta x)(y - \Delta y) = xy$ 라는 공식이 만들어 진다.
4. Input 값인 Δx 는 사용자가 입력한 값이므로 정해져있고, Output 값인 Δy 를 구하는 것이 핵심이다.
5. 공식을 분해해보면 아래와 같이 나온다.

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

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

$$\Delta xy = x\Delta y + \Delta x\Delta y$$

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


6. 최종적으로 $\Delta y = \frac{y\Delta x}{x + \Delta x}$ 가 나온다.

／

／

CPMM

```
function getPrice(uint256 inputReserve, uint256 outputReserve) public pure returns (uint256) {  
    uint256 numerator = inputReserve;  
    uint256 denominator = outputReserve;  
    return numerator / denominator;  
}
```

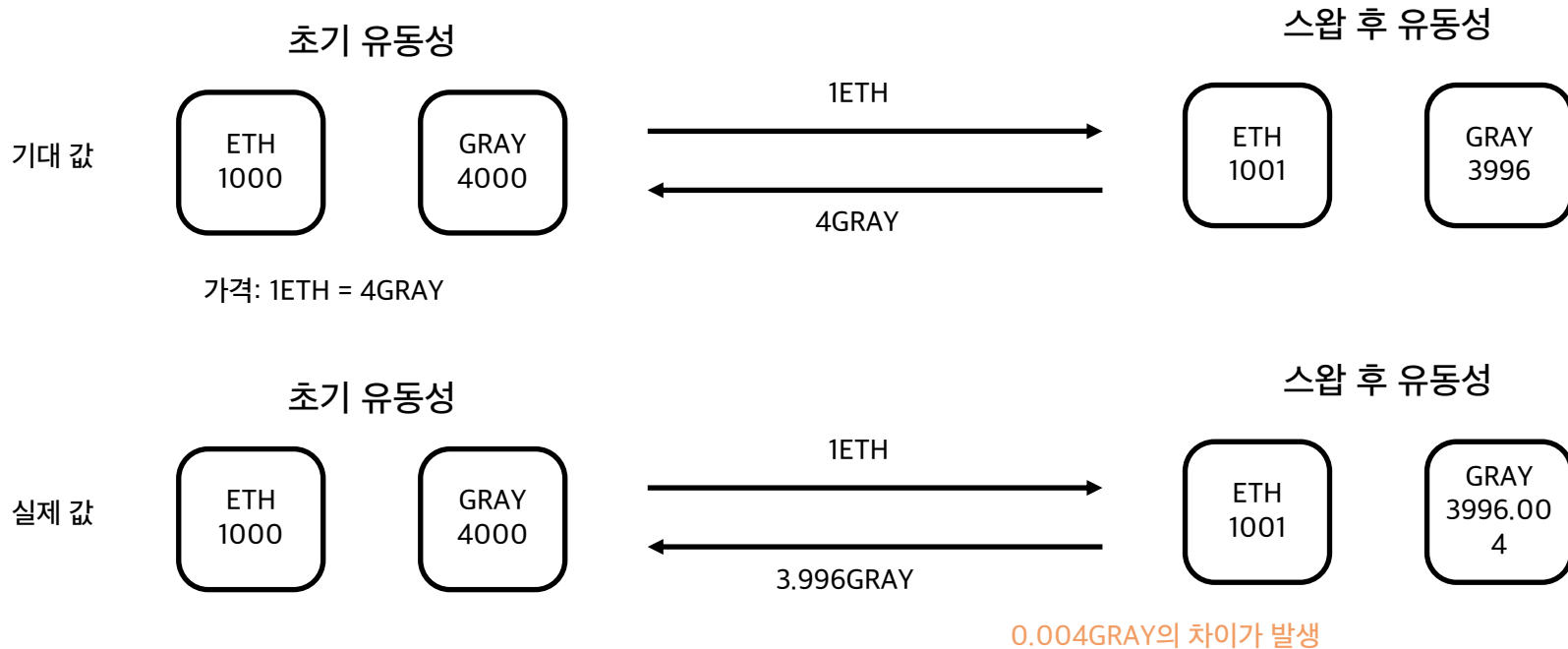

$$\Delta y = \frac{y\Delta x}{x + \Delta x}$$

```
function getOutputAmount(uint256 inputAmount, uint256 inputReserve, uint256 outputReserve)  
    uint256 numerator = (inputAmount * outputReserve);  
    uint256 denominator = (inputReserve + inputAmount);  
    return numerator / denominator;  
}
```

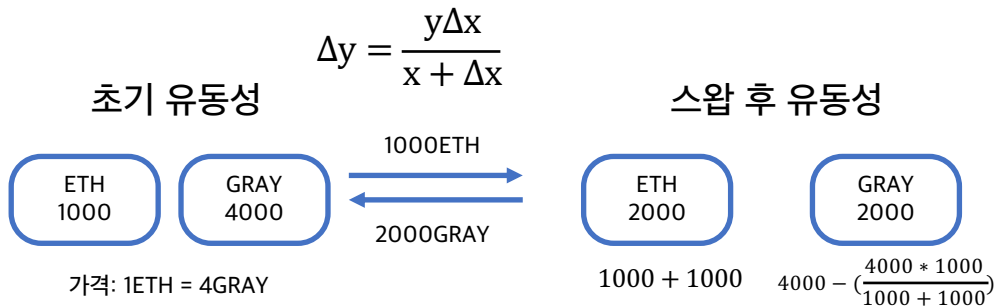
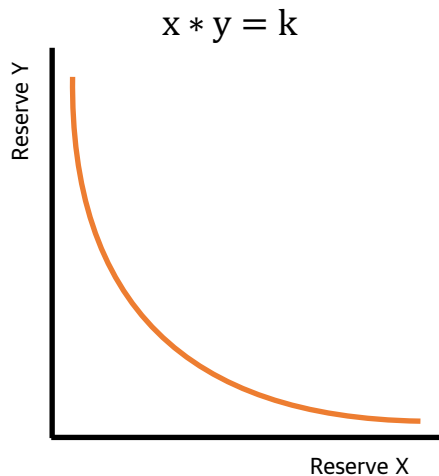
구현 및 테스트(CPMM)

git clone https://github.com/GrayWorld-io/lec_fc_defi

Detail Swap



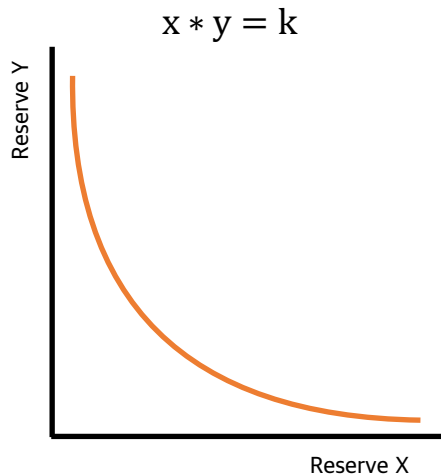
Detail Swap



```
expect(  
  toEther((await exchange.getOutputAmount(toWei(1000000), etherReserve, tokenReserve)))  
).to.eq("3996.003996003996003996");
```

극단적으로 10만ETH를 스왑해도 풀에있는 4000개의 토큰을 모두 가져오지 못한다.

슬리피지



- 슬리피지는 스왑을 통해 받게 되는 기대 토큰 개수와 실제 받는 토큰 개수와의 차이이다.
- CPMM에서는 풀에 존재하는 토큰의 양에 따라 받게 되는 토큰의 양이 정해지므로 필연적으로 발생 할 수 밖에 없다.
- 풀에 존재하는 토큰은 0이 되지 않는다. 즉 유동성은 고갈되지 않는다.
 - 이것이 가능한 것도 슬리피지가 있기 때문이다.
- 같은 가격이어도 유동성 양에 따라서 슬리피지는 달라진다.
 - 유동성이 클수록(k값이 클수록) 슬리피지는 작아진다.
- 적은 양을 교환하는 것이 슬리피지를 줄일 수 있다.
- 디파이 서비스들은 슬리피지를 줄이기 위해 유동성을 최대한 끌어모으려고 한다.

다음 강의

이번에 구현한 가격측정 함수를 활용하여 Swap을 구현한다.