Building Uniswap V2 Simulation

To begin, I'd like to go over the basic logic behind the way I designed the project. I chose to create a poolStruct Class, which allows me to create instances of pools that I can simulate trades within as well as the process of spotting arbitrage opportunities. Further, a Trader class models (in java) the logic behind when to calculate whether or not an arbitrage opportunity exists, and if it does, makes the optimal trade. The TLDR summary of all classes is here:

poolStruct: Object that allows us to instantiate pool objects, and contains methods that allow adding, removing, and trading tokens.

Trader: Object that models the logic that a trader would have in an environment where a trader is looking at two pools.

SimulationEnv: Contains executable main method that runs through an example scenario and prints out the states of the pools at each step

Within the Pool object, there are (effectively) three methods.

Basic Struct Outline: PoolStruct

Saved Data:

The struct keeps track of the current reserves of Dai, Eth, as well as the outstanding LP tokens and the swap fee implemented with it (which in our case will always be 0.3%)

Add:

For a given input of Dai and Eth, we will add the Dai and Eth to the poolStruct object, increasing the liquidity within the pool. We then will reward/return LP tokens to the user who is pooling their tokens. More specifically, we will reward the user exactly $\sqrt{d^2 + e^2}$ LP tokens, where d is the quantity of Dai, and e is the quantity of Eth inputted. Additionally, I accounted for the edge case where a user might try to "game the system" by adding improper ratios of d and e that do not align with the current pool ratio by implementing the rules in the Uniswap protocol that punish this behavior by causing the user to systematically receive less LP tokens when the ratio of e and/or d is not equal to that of the pool.

Remove:

When a user wants to remove their LP tokens from the pool, they get rewarded by (potentially) receiving more d and e than what they put in – depending on how long they provided liquidity for. For a user with x LP tokens where there are x_{tot} outstanding LP tokens,

they will receive $\frac{x}{x_{tot}} \times d_{tot} = d$ and $\frac{x}{x_{tot}} \times e_{tot} = e$. Where e_{tot} and d_{tot} are the total Dai and Eth in the pool respectively, and e and d are the Etherium and Dai received in exchange for the x LP tokens.

Trade:

In the explanation of this method I'll assume WLOG we're trading Dai to Eth. For an input quantity d_{in} of Dai, we first will consider the quantity after the fee is taken off, which gives $d_{fee} = d_{in} \times 0.997$ then, we know that our updated quantity of dai will be $d_{new} = d + d_{fee}$ where d is the quantity of Dai in the pool. From this, we can calculate what the new value of Eth must be in order to maintain the invariant of k for $d \times e = k$.
Using this, we get that $e_{new} = \frac{d \times e}{d_{new}}$ must be the new value of Ethereum in order to preserve the constant k. Finally, the difference between the old value of e and the new value of e gives us the amount of dai received in the swap: $receievedEth = e - e_{new}$
Where k is maintained, the fee is applied, and the trader receives Eth in exchange for Dai. An Edge case here is accounted for, which is that in certain cases a trader may try to buy more Eth or Dai than the exchange can produce, in which case we'd see that (in our previous example) $receievedEth = e - e_{new} < 0$ meaning that we cannot provide liquidity in this case and we (for the purposes of the simulation) simply throw an exception.

Defining Arbitrage Opportunities

With the basic structure of how transactions and Liquidity works in this market in place with the Class described above, the next step was to mathematically define when arbitrage was possible, and what the maximum profit was in terms of the values of Eth and Dai in each of the two Pools. For this simulation, we're only interested in arbitrage between two pools.

Factually, positive valued arbitrage can only exist between two pools when, for $Pool_1$ with E_1 and D_1 , and $Pool_2$ with E_2 and D_2 reserve balances of Eth and Dai respectively, have the property that $\frac{D_1}{E_1} \neq \frac{D_2}{E_2}$. For the following explanation of the scenarios in which arbitrage opportunities exist and for what maximum profit, I'll assume that $\frac{D_1}{E_1} > \frac{D_2}{E_2}$, where for the opposite case we'd simply switch 1 and 2. For $\frac{D_1}{E_1} > \frac{D_2}{E_2}$, the logical explanation is that a trader would buy Dai with Eth at the $\frac{D_1}{E_1}$ exchange rate, and sell at the $\frac{D_2}{E_2}$ exchange rate for a profit, but we must know (1) how big this gap must be for profit to occur (accounting for fees) and (2) when a big enough gap exists, what the optimal input quantity of Eth is.

We can model the exchange rate for E_1 to D_1 in $Pool_1$ as:

$$D_{out} = D_1 - \left[(D_1 \times E_1) / (E_1 + (E_{input} \times 0.997))\right]$$

And further the exchange rate for D_2 to E_2 in $Pool_2$ as:

$$E_{out} = E_2 - \left[(E_2 \times D_2) / (D_2 + (D_{input} \times 0.997))\right]$$

Which tells us that for an input quantity of Etherium E_{input} into $Pool_1$ we can end up with

$$E_{out} = E_2 - \left[(E_2 \times D_2) / (D_2 + (D_1 - \left[(D_1 \times E_1) / (E_1 + (E_{input} \times 0.997))\right]) \times 0.997))\right]$$

Where more precisely, our profit will be

$$Profit = E_{out} - E_{input}$$
$$Profit = \left[E_2 - \left[(E_2 \times D_2) / (D_2 + (D_1 - \left[(D_1 \times E_1) / (E_1 + (E_{input} \times 0.997))\right]) \times 0.997))\right]\right] - E_{input}$$

Using this, we know the goal is to maximize profit, and we can find the input value of Eth which does this by deriving our profit equation with respect to E_{input} and solving for the zeros

$$\frac{\partial Profit}{\partial E_{input}} = \frac{994009 E_1 E_2 D_1 D_2}{1000000 \left(\frac{997\left(D_1 - \frac{E_1 D_1}{0.997 E_{\text{input}} + E_1}\right)}{1000} + D_2 \right)^2 \cdot (0.997 E_{\text{input}} + E_1)^2}$$

$$\frac{\partial Profit}{\partial E_{input}} = 0 \Rightarrow E_{input} = \frac{997000\sqrt{E_1 E_2 D_1 D_2} - 1000000 E_1 D_2}{997000 D_2 + 994009 D_1}$$

So, we've defined the E_{input} that maximizes our Profit for a given combination of reserve balances of two pools, and we know that some form of arbitrage exists for $\frac{D_1}{E_1} \neq \frac{D_2}{E_2}$, but we also know that for $\frac{D_1}{E_1} \neq \frac{D_2}{E_2}$, when they are extremely close, a possible arbitrage opportunity may not

yield positive profit because of the 0.3% fee. However, we can use the fact that we know the E_{input} that maximizes profit to check if

$$E_{out} - E_{input} > 0 \equiv Profit > 0$$

$$\equiv \left[E_2 - \left[(E_2 \times D_2) / (D_2 + (D_1 - \left[(D_1 \times E_1) / (E_1 + (E_{input} \times 0.997))\right]) \times 0.997))\right]\right] - E_{input} > 0$$

For $E_{input} =$ $$\frac{997000\sqrt{E_1 E_2 D_1 D_2} - 1000000 E_1 D_2}{997000 D_2 + 994009 D_1}$$

Then the trade when executed with optimal E_{input} must yield positive profit, and we'd execute the Arbitrage trade and exchange E_{input} Etherium for D_1 in $Pool_1$ and Sell the resulting Dai in $Pool_2$ to get E_{out}, which we know $E_{out} - E_{input} > 0$ yielding positive profit. For the scenario in which $\frac{D_1}{E_1} \neq \frac{D_2}{E_2}$ and $\frac{D_1}{E_1} < \frac{D_2}{E_2}$, the above holds true when 1 and 2 are replaced.

When Should Arbitrage Calculations Be Triggered?

I'm not highly familiar with non-blocking mode, but from research, I can imagine we want to trigger such an arbitrage calculation process as little as possible, and as efficiently as possible. When should we do this? From the implementation of the Uniswap V2 protection against adding Eth Dai pairs in a way that offsets the current Eth Dai exchange rate of such a pool, the only times we want to trigger a check to see if an arbitrage opportunity exists is when (1) the first amounts of Eth and Dai are added to a new pool or (2) a trader buys/sells Eth or Dai from a pool (which could change the exchange rate of Eth/Dai in that pool, tipping it over the edge of profitability). In this case, each time a trade occurs a trader would want to examine whether the resulting reserve balances in the two pools change in a way that yields arbitrage opportunity, then execute the trade. Additionally, from the math above, if an arbitrage trade is triggered, we will not check again until one of the above conditions is met, because when the optimal arbitrage trade is executed, the market will get changed to a spread that is not profitable.

Basic Testing Logic

To give an overview of how my implementation works holistically, walking through a testing procedure could be useful. Consider the case where we have pool1 with 200 Dai and 50

Eth, and pool2 with 150Dai and 50 Eth: ie. $D_1 = 200$, $D_2 = 150$, $E_1 = 50$, $E_2 = 50$. In the testing case (and in most testing cases) the trader that exists only checks for arbitrage operations for the two cases above (in this case the trader checks for an arbitrage operation when both pools get filled for the first time) so as to mimic the real world where these calculations must occur in non-blocking mode. We want to test that the trader (1) identifies the arbitrage opportunity, and (2) chooses the correct amount of eth. In this case, the trader chooses the value ~3.256, which we can double-check by graphing the input Eth purchase size on the x-axis, and the resulting profit on the y-axis (seen below), which results in the same answer. This basic example gives an idea for (1) the reasoning behind the choice of optimal eth quantity, (2) testing when the trader will utilize a check operation to check for arbitrage opportunity, and (3) show that the trader will make a profit.