17 JANUARY 2022

GREEN ART TOKEN

DECENTRALISED EXCHANGE

Alexei Goloubtchikov
BLOCKCHAIN ENGINEER

LIQUIDITY POOL

Green art token decentralised exchange (DEX) applies Automatic Market Maker (AMM) model to provide liquidity, establish fair prices for GAC/ETH (or other quote currency) pair trading and commission rewarding to market makers. One part of accumulated commissions will be mandatory sent to charity wallet(s); all trading and price generation logics is written in smart contract itself and can be independently controlled by auditors. Here we describe DEX realisation challenges and our choices to realise DEX logics using ETH as quote currency; using other networks such as BSC (Binance Smart Chain), Fantom, Polygon or any layer 2 networks require to substitute ETH in text below to native network coin.

CONSTANT RATIO OF TRADED PAIRS

Green Art Coin DEX (Exchange) is based on the constant ratio of traded pair formula:

$$x * y = k$$

Where **x** is the ETH reserve, **y** is the token reserve (or vice versa), and **k** is a constant. Green Art Coin DEX requires **k** to remain the same no matter how many **x** or **y** reserves exist. When you exchange ETH for tokens, you put your ETH into a smart contract and receive some tokens in return. Green Art Coin DEX guarantees that after each trade, **k** remains the same.

PRICING FUNCTION

How are we going to calculate stock prices?

It may be tempting to think that price is simply a ratio of stocks, for example:

$$P_X = \frac{y}{x}, P_Y = \frac{x}{y}$$

This is logical: Exchange smart contracts do not interact with centralised exchanges or any other external price oracles, so they cannot know the correct price. In fact, the exchange smart contract is a price oracle in itself. All they know is the stock of ETH and GAC tokens, and that's the only information they have for pricing.

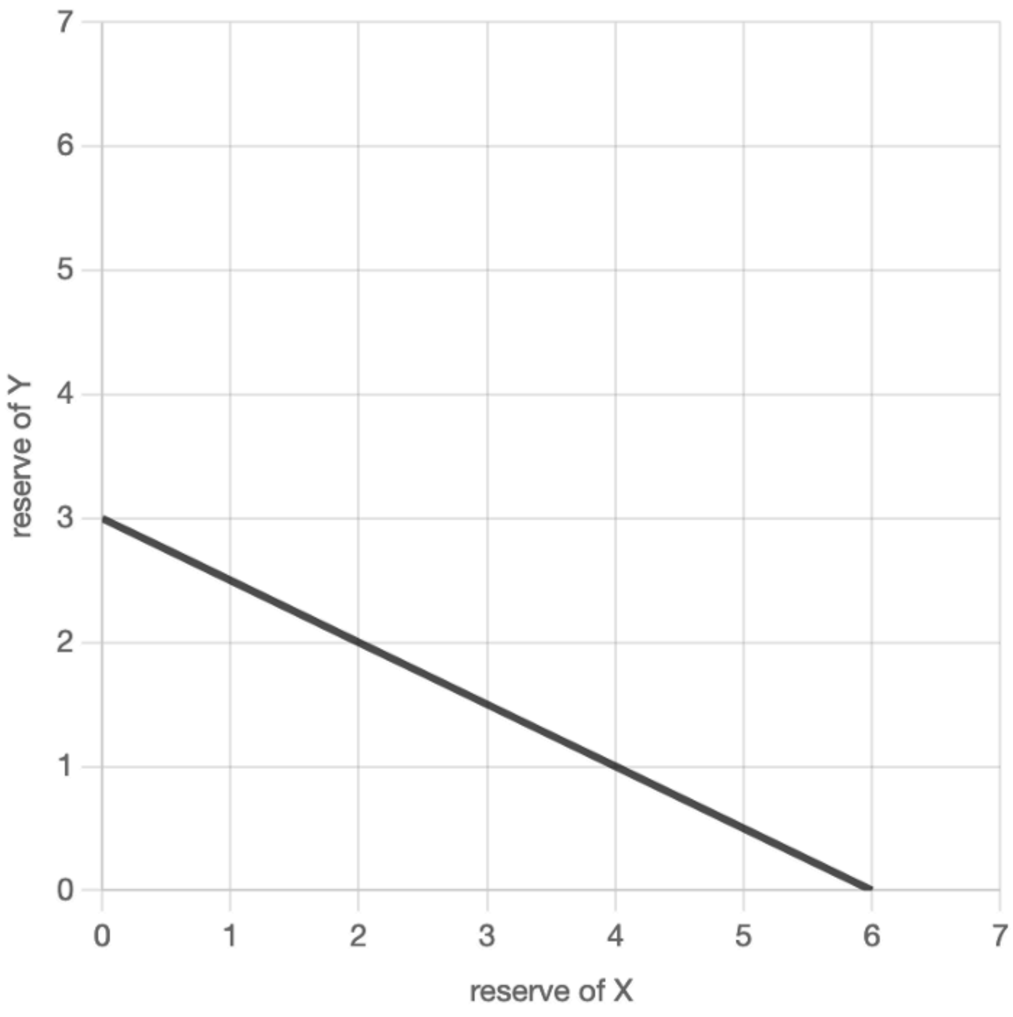
Let's stick with this idea and build a pricing function:

```
function getPrice(uint256 inputReserve, uint256 outputReserve)
public pure returns (uint256) {
  require(inputReserve > 0 && outputReserve > 0, "invalid
reserves");
  return inputReserve / outputReserve;
}
```

We have deposited 2000 GAC tokens and 1000 ETH and expect the token price to be 0.5 ETH and the price of ETH to be 2 tokens. That's all right.

But what happens if we exchange 2000 GAC tokens for ETH? We will receive 1000 ETH, which is all we have under the smart contract! The stock exchange will be emptied!

The reason for this is that the pricing function belongs to the constant sum formula, which defines **k** as a constant sum of **x** and **y**. The function of this formula is a straight line and cross both x and y axis:



PROPER PRICING FUNCTION

Green Art Coin DEX is a constant ratio traded pairs automatic market maker, which means that it is based on the constant ratio traded pairs formula:

$$x * y = k$$

The formula says that **k** remains constant no matter what the reserves are (**x** and **y**). Each trade increases the ETH or GAC token reserve and decreases the GAC token or ETH reserve.

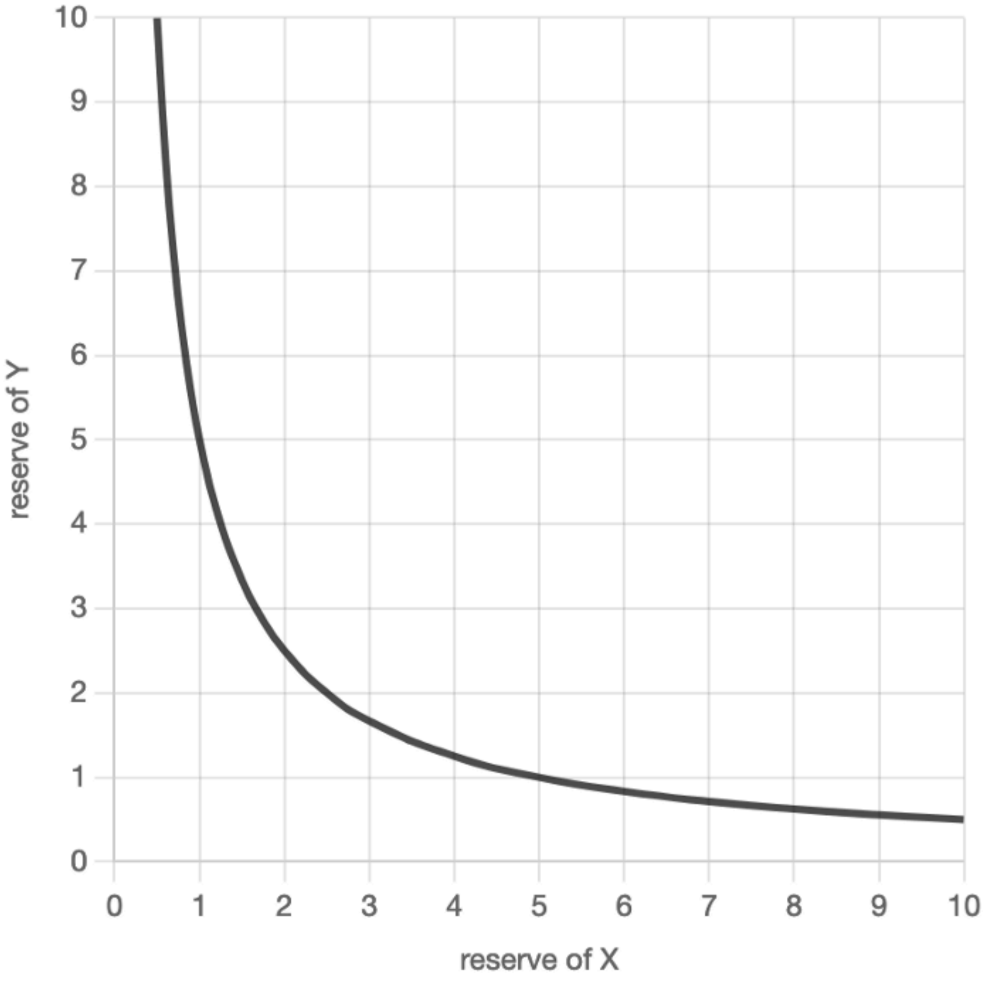
Let's put this logic into a formula:

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

where **Δx** is the amount of ETH or GAC tokens we are exchanging for and **Δy** is the amount of GAC tokens or ETH we are getting in exchange. Using this formula, we can now find **Δy**:

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

The formula for the constant ratio of traded pairs that we based our price calculations on is actually a hyperbole:



The hyperbola never crosses **x** or **y**, so none of the reserves are ever 0. This makes the reserves infinite.

This pricing function causes “price slippage”. The greater the number of tokens traded in relation to reserves, the lower the price will be.

This is consistent with the law of supply and demand: the higher the demand (the more you want) relative to the supply (reserves), the higher the price (the less you get).

Also, this is the mechanism that protects pools from being emptied.

ADDING LIQUIDITY

We must ensure that the added liquidity is placed in the same proportion of eth-token that is already established in the pool. At the same time, we want that at the stage of creating the exchange, liquidity will flow in an arbitrary proportion of eth-token, when the initial reserves are empty, i.e. when the pool has not yet been initialized. This is important because it is at this point that the price of the eth token is set for the first time.

Add liquidity function has two branches:

- if this is a new exchange (no liquidity, pool is empty), allow an arbitrary amount of liquidity.

- Otherwise, keep the prescribed proportion of reserves when there is liquidity.

```
if (getReserve() == 0) {
    IERC20 token = IERC20(tokenAddress);
    token.transferFrom(msg.sender, address(this), _tokenAmount);
} else {
    uint256 ethReserve = address(this).balance - msg.value;
    uint256 tokenReserve = getReserve();
    uint256 tokenAmount = (msg.value * tokenReserve) / ethReserve;
    require(_tokenAmount >= tokenAmount, "insufficient token amount");
    IERC20 token = IERC20(tokenAddress);
    token.transferFrom(msg.sender, address(this), tokenAmount);
}
```

We do not deposit all tokens provided by the user, but only the amount calculated based on the current reserve ratio. To get the amount, we multiply the ratio (tokenReserve / ethReserve) by the amount of ETH deposited. If the user has invested less than this amount, an error will be thrown.

This will save the price while adding liquidity to the pool.

We need to have a way to reward liquidity providers for their contributions. If they are not motivated, then they will not provide liquidity, because no one will invest their ETH

/ GAC in a third-party smart contract just like that. Moreover, liquidity providers should not be rewarded by us, because for this we would have to look for investments or issue our own inflationary token.

This will save the price while adding liquidity to the pool.

The only good solution is to charge a small fee on each token exchange and distribute the accumulated fee among the liquidity providers. **Part of this commission will be sent to the charity wallet!**

To be rewarded fairly, we must reward liquidity providers in proportion to their contribution, i.e. the amount of liquidity they provide. There is an elegant solution: liquidity provider tokens or LP tokens.

LP tokens are essentially ERC20 tokens automatically issued and transferred to liquidity providers in exchange for their contribution to liquidity. In essence, LP tokens are shares:

1. You receive LP tokens in exchange for your liquidity that you provided to the pool.

2 .The number of LP tokens you receive is proportional to your share of liquidity in the pool reserves.

3. Commissions are distributed in proportion to the number of LP tokens you own.

4. LP tokens can be exchanged back for liquidity and receive accumulated commissions (minus commissions sent to charity wallets)

How will we calculate the number of issued LP tokens depending on the amount of liquidity provided? It's not so obvious because there are some requirements that we need to meet:

1. Each issued share must always be correct. If someone after me adds or removes liquidity from the pool, my share must remain equal to my contribution to the total liquidity.

2. Write operations (for example, saving new data or updating existing data in a smart contract) are very expensive. Therefore, we want to reduce the cost of maintaining LP tokens

Let's suppose that we issue a lot of tokens (1 million) and distribute them among all liquidity providers. If we always distribute all the tokens (the first liquidity provider receives 1 million, the second - his share, etc.), then we are forced to recalculate the issued shares later, which is expensive. If we initially distribute only a part of the tokens, then we run the risk of hitting the supply limit, which will eventually force us to redistribute the existing shares.

The only good solution is the complete absence of limits on the supply and release of new tokens when adding new liquidity. This allows LP tokens to grow indefinitely, and if we use the correct formula, all issued LP tokens will remain in the correct ratio to the total amount of liquidity (will proportionally increase) when liquidity is added or removed.

LP TOKENS

How to calculate the number of LP tokens to be issued when depositing liquidity?

Reserves of ETH and tokens are stored in the smart contract of the DEX. But how will we calculate the number of LP tokens? Based on the general reserve? Or just one of them (ETH, GAC)? We calculate the number of LP tokens in proportion to the reserve of ETH:

$$amountMinted = totalAmount * \frac{ethDeposited}{ethReserve}$$

When adding initial liquidity, the number of issued LP tokens is equal to the number of deposited ETH. Additional liquidity issues LP tokens in proportion to the amount of ETH invested:

```
function addLiquidity (uint256 _tokenAmount)
    public
    payable
    returns (uint256)
{
    if (getReserve() == 0) {

        uint256 liquidity = address(this).balance;
        _mint(msg.sender, liquidity);
        return liquidity;
} else {

        uint256 liquidity = (totalSupply() * msg.value) / ethReserve;
        _mint(msg.sender, liquidity);
        return liquidity;
    }
}
```

COMISSIONS

We need to answer a few questions:

1. Do we want to take commissions in ETH or GAC? Do we want to reward liquidity providers in ETH or GAC?

2. How to collect a small fixed fee from each exchange?

3. How to distribute the accumulated commissions between liquidity providers in proportion to their contribution?

How to solve this?

We may establish an additional payment that is sent along with the exchange transaction. Such payments are accumulated in a fund from which any liquidity provider can withdraw an amount proportional to their share (amount is bit less that accumulated quantity because of mandatory charity wallet transaction requirement)

We already have all we need for this scenario.

1. Traders are already sending ETH/GAC to the exchange smart contract. Instead of asking for a fee, we can simply subtract it from the ETH/GAC that are sent to the smart contract.

2. We already have a fund - these are the reserves of the Exchange. Reserves can be used for accumulated fees. This also means that reserves will grow over time.

3. Commissions are paid in the currency of the asset being traded. Liquidity providers receive a balanced amount of ETH and GAC plus a share of accumulated fees proportional to the share of their LP tokens.