BLOK8 / Swappy Private

A Decentralized Exchange, for instant swap of tokens

0 stars 0 forks

Star Watch

Code Issues Pull requests 1 Actions Projects Wiki Security Insights

develop

This branch is even with master.

#70 Compare

AkshayCHD ... on Jun 15

View code

README.md

Swappy

A Decentralized Exchange, for instant swap of tokens

Overview

Swappy is a decentralized exchange for exchanging ERC20 tokens on the ethereum blockchain. A team of 3 Developer Interns, 2 QA Interns, 1 Agile Delivery Manager, 1 QA Supervisor and a Project Lead have been assigned this project to work on. The project is built on the ethereum blockchain where the smart contracts would be programmed in solidity.

Problem Statement

In the ethereum ecosystem, there is a specific standard for implementing fungible tokens, that standard is known as ERC20 token, and since this standard came into picture, the number of ERC20 compliant tokens have flooded the Ethererum Ecosystem, but this gave arise to need to exchange these tokens. But the traditional centralized exchanges make use of order books for 2 ERC20 tokens, which have a lot of drawbacks, such as the time taken in completing an exchange and the high transaction fees involved. Also these exchanges have a lot of terminologies that the user has to get himself acquainted to, before making the exchange which becomes a daunting task for new users.

About Swappy

Swapi is a completely decentralized exchange, i.e all its logic is entirely deployed to the ethereum blockchain, which allows any exchange that is being performed on the platform to be executed by smart contracts, which eliminates the need for trusted third party verification and makes the exchanges instantly. Also, instead of using the traditional order books as was the case with centralized exchanges, Swapi uses an exchange rate algorithm to determine the exchange rate for one token to another. Last but not least, because of the above two factors the number of entities involved in an exchange are very less, hence the transaction charge is also significantly lower than the centralized exchanges.

Salient Features

- The platform will allow users to instantly exchange one ERC20 token for another, with very low transaction charges involved.
- Users can create an exchange for any arbitrary ERC20 pair.
- Users can add liquidity to earn interest on it, the added liquidity can be withdrawn at any time.
- Each time a token exchange takes place on the platform 0.03% of the input amount is taken up as transaction charge and is distributed proportionally among all the liquidity providers.

Architecture

Factory Contract

Factory contract is the center of the entire Swapi architecture, the contract is responsible for creating pair contracts on runtime.

Pair Contracts

In Swapi, each ERC20 token pair whose liquidity exists on the platform, there is a dedicated pair contract, specifically for that token pair. The pair contract is responsible for the management of reserves of both types of token. Apart from that each pair contract is an ERC20 token itself, i.e for a pair each pair contract we are dealing with three types of tokens, out of which two are the tokens whose exchange the pair contract represents (referred to as reserve tokens) and the third is the pair's intrinsic token. On a broader level pair contract offers the following functionalities:

Mint: If the contract holds an excess value of the reserve tokens, take note of that excess amount and mint an equivalent amount of the pair's intrinsic token to address provided in the parameter. Burn: If the contract holds an excess amount of pair's intrinsic tokens, then burn those and provide an equivalent amount of reserve tokens to the address provided in the parameter. DrawTokens: Draw any amount of reserve tokens from the contract, making sure that the product of the amount of reserve tokens held by the contract does not decrease.

Router Contract

In Swapi, the pair contract does not employ any means to safeguard the funds that are provided by the user to perform the exchange, the only thing that the pair contract is concerned with is the safety of the reserves of tokens it has, so that no user can draw more value from the contract than he is allowed to, due to which the consistency of the reserves is maintained at all times. The safety of the resources provided by the user is made sure by the router contract. The router contract performs all kinds of safety checks, so as to safeguard the resources provided by the user, and provide him with the most optimal value in return for the resources provided.

Exchange Rates Algorithm

The principle on which the algorithm works is: After any exchange of tokens the product of reserves of the two tokens must not change, i.e

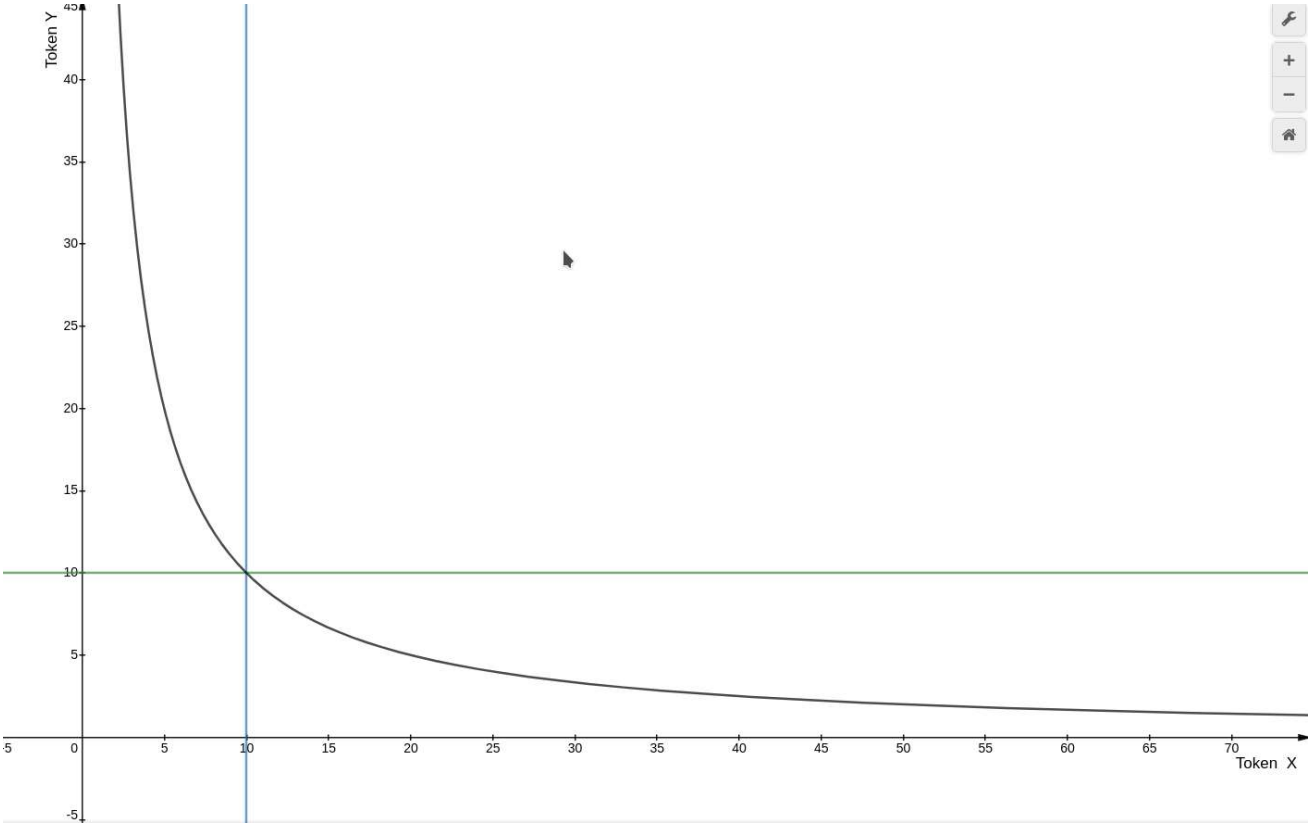
```
X * Y = K
```

Where is K is known as an invariant, and X and Y are the amount of tokens in the system

For example: Let there be 2 tokens X and Y and the value of their reserves be 10 and 10 respectively. So as per our property of invariant the equations becomes

```
X * Y = 100
```

Which can be represented by the following graph,



In the graph above the black line represents the relation

```
x * y = 100
```

And the blue and green lines represent the relations

```
x = t, y = k / t
And currently t = 10
```

Now let us say we want to exchange some amount of y tokens to get 5 tokens of type x; So, in this case the exchange has to performed in such a way that the property of invariant remains intact, i.e

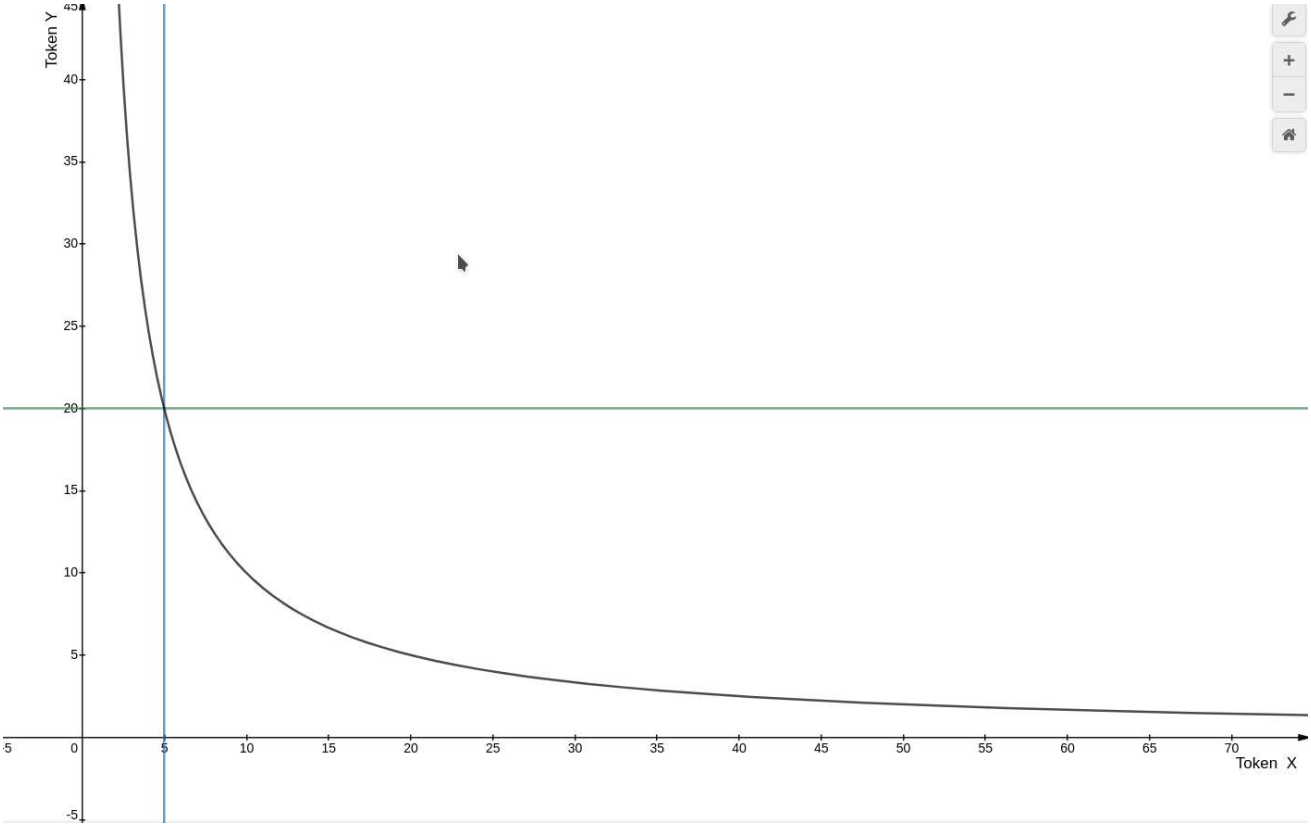
```
x * y = 100
```

After the exchange.

So we can deduce that the value of t after the exchange will become

```
t = 10 - 5 = 5
```

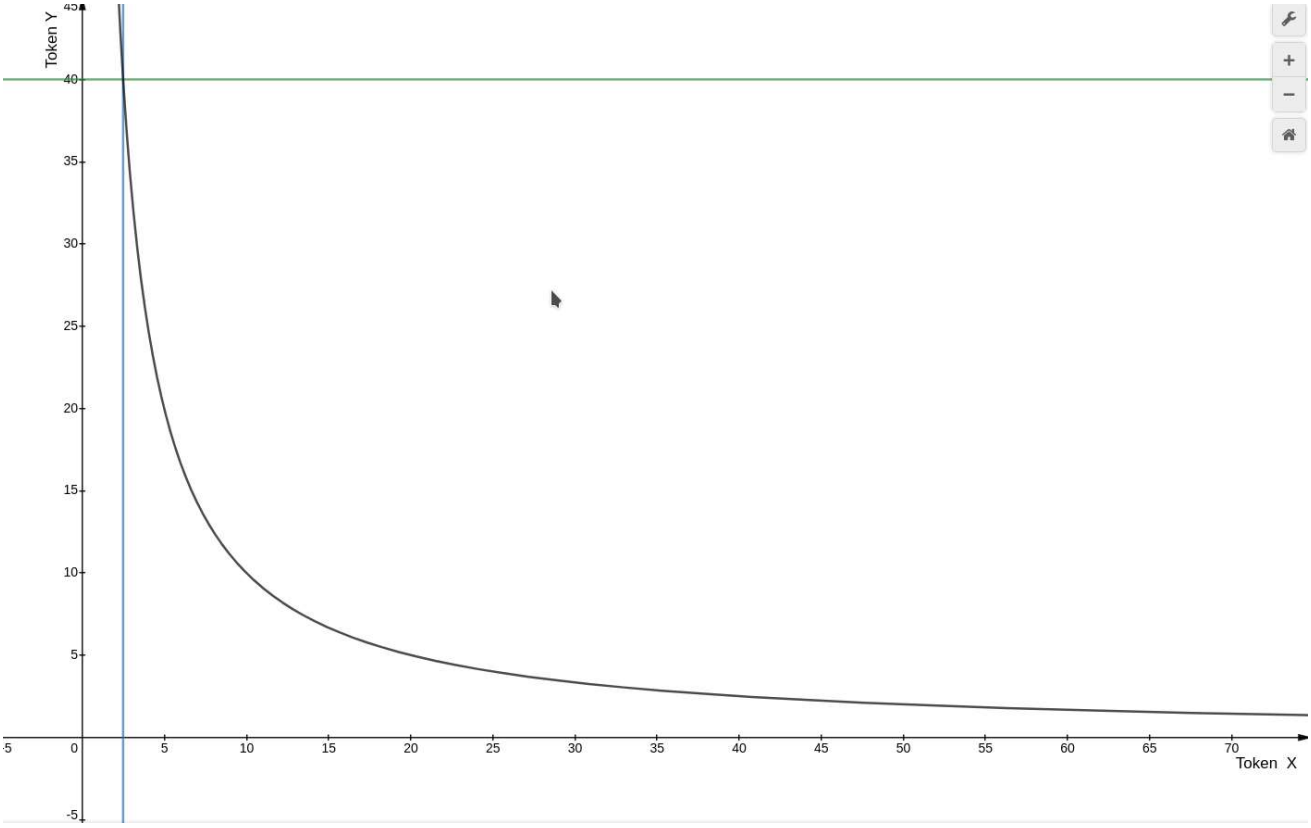
So the new graph will be something like



We observe that the value of y after the exchange will have to be 20 to satisfy the property of invariant, that means to get 5 tokens of x type we had to give 10 tokens of y type. Furthermore if we again want to get some x type tokens, let us say x = 2.5 this time, the value of t will be

```
t = 5 - 2.5 = 2.5
```

The following graph represents the values of x and y after the above exchange is performed.



So this time the value of y will be 40, i.e the number of y tokens we will have to provide will be

```
y = 40 - 20 = 20
```

So we observe the value of x tokens in terms of y tokens is increasing as the demand for x tokens is increasing, so the property of invariant very clearly mimics the impact of supply and demand on a traditional exchange and hence this property is used to determine the exchange rates in case of Swapi.

Exchange Rate Formula

Let us suppose there are two tokens in the system ABC and DEF, with initial value of reserves as A and D respectively. We want to exchange `a` amount of ABC tokens for any amount of DEF tokens, let's say `d`. So, our aim is to calculate the value `d` that we will get in return of `a` tokens provided. As per the principle of invariant, we can say

```
A * D = (A + a)(D - d)
```

We can solve this equation in the following manner to get the value of d.

```
A * D = (A + a)(D - d)
```

Taking the (A + a) from RHS to LHS as denominator and reversing the eq.

```
D - d = (A * D)(A + a)
(-d) = ((A * D) / (A + a)) - D
```

Taking value of D common

```
d = D(1 - A / (A + a))

d = D((A + a - A) / (A + a))

d = (D * a) / (A + a)
```

Amount of d as Output.

Exchange Fees

A fee of 0.03% of the input tokens is charged for each exchange performed on the platform, and is distributed proportionally amongst all the liquidity providers on the basis of the amount of liquidity provided by them. So the effective formula for calculation on the number of output tokens (d) becomes,

```
d = (D * p) / (A + p)
where p = a * 0.997
```

Liquidity Providers

For the system to actually be able to swap tokens one for another, the system should have a reserve of tokens, that reserve is provided by liquidity providers. On the basis of the ratio of liquidity provided by them, they recieve an equivalent proportion of the 0.03% fee charged on each exchange transaction. For example: If someone's share in the liquidity pool is 25% then he will get 25% of the 0.03% of each transaction fees.

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 3


AkshayCHD Akshay Kumar


Sahil1109


RaviBlock8 RaviVerma

Languages

- **JavaScript** 100.0%